TP Synthèse d'images: Modélisation
CPE

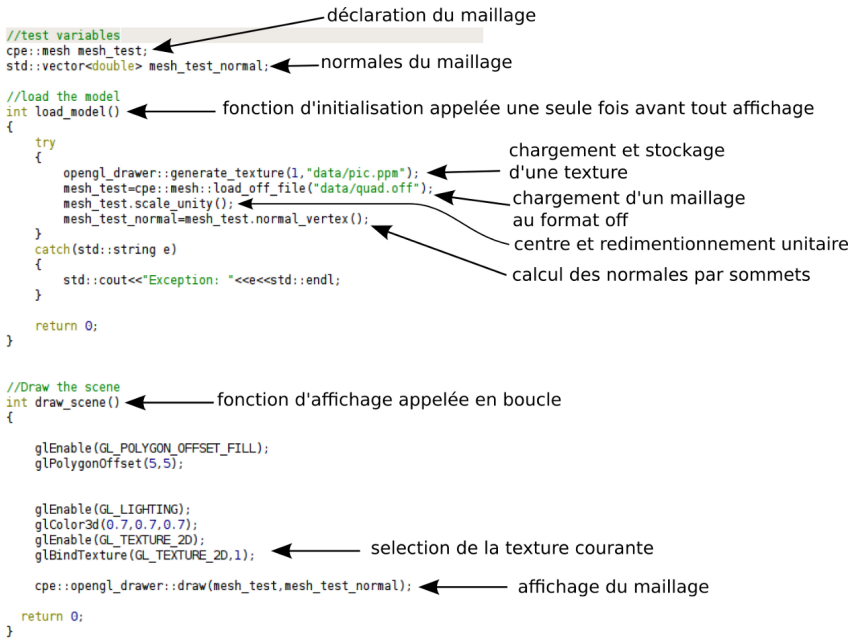
durée - 4h

03 Decembre 2010

1 Prise en main de l'environnement

1.1 Fonctions du main

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




```
//test variables
cpe::mesh mesh_test;
std::vector<double> mesh_test_normal;

//load the model
int load_model()
{
    try
    {
        opengl_drawer::generate_texture(1,"data/pic.ppm");
        mesh_test=cpe::mesh::load_off_file("data/quad.off");
        mesh_test.scale_unity();
        mesh_test_normal=mesh_test.normal_vertex();
    }
    catch(std::string e)
    {
        std::cout<<"Exception: "<<e<<std::endl;
    }

    return 0;
}


//Draw the scene
int draw_scene()
{

    glEnable(GL_POLYGON_OFFSET_FILL);
    glPolygonOffset(5,5);


    glEnable(GL_LIGHTING);
    glColor3d(0.7,0.7,0.7);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D,1);

    cpe::opengl_drawer::draw(mesh_test,mesh_test_normal);

  return 0;
}
```

FIGURE 1 – Explication de l'affichage initiale.

Question 1 *Effectuez le chargement d'un maillage complexe sous le modèle de la fig. 2.*

```
//test variables
cpe::mesh mesh_test;
std::vector<double> mesh_test_normal;

//load the model
int load_model()
{
    try
    {
        mesh_test=cpe::mesh::load_off_file("data/dino_1.off");
        mesh_test.scale_unity();
        mesh_test_normal=mesh_test.normal_polygon();
    }
    catch(std::string e)
    {
        std::cout<<"Exception: "<<e<<std::endl;
    }

    return 0;
}


//Draw the scene
int draw_scene()
{

    glEnable(GL_POLYGON_OFFSET_FILL);
    glPolygonOffset(5,5);


    glEnable(GL_LIGHTING);
    glColor3d(0.7,0.7,0.7);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D,1);

    cpe::opengl_drawer::draw(mesh_test,mesh_test_normal);

    glDisable(GL_POLYGON_OFFSET_FILL);
    glColor3d(0,0,0);
    cpe::opengl_drawer::draw(mesh_test);

  return 0;
}
```

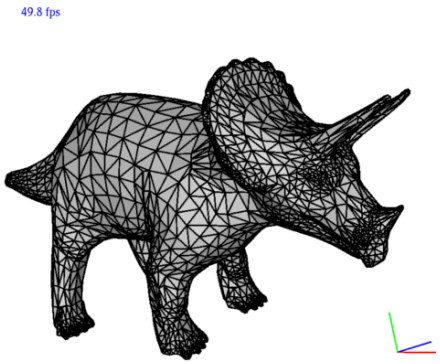


FIGURE 2 – Affichage d'un maillage complexe.

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 disponible 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*.
La fonction *load model* est appelée une unique fois au début du programme.

Question 2 *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. 3 (avec sa bordure) :*

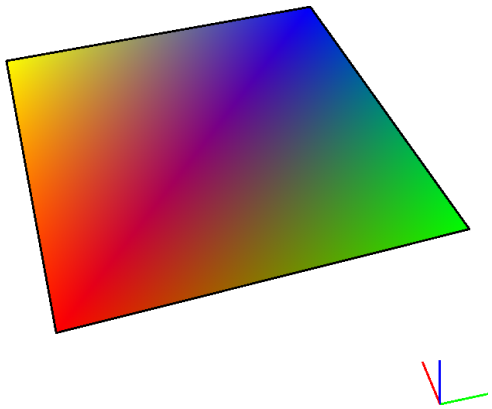


FIGURE 3 – Affichage d'un quad.

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 a 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 tetraedre est proposé en fig. 4.

```
//Draw the scene
int draw_scene()
{

    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());

    return 0;
}
```

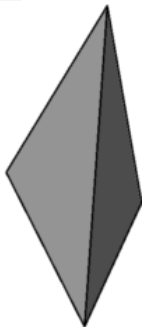


FIGURE 4 – Affichage d'un tetraedre a l'aide de la class *mesh*.

2 Cylindre généralisé

Un cylindre généralisé consiste à extuder un cylindre le long d'une courbe donnée. Un exemple est fourni en fig. 5.

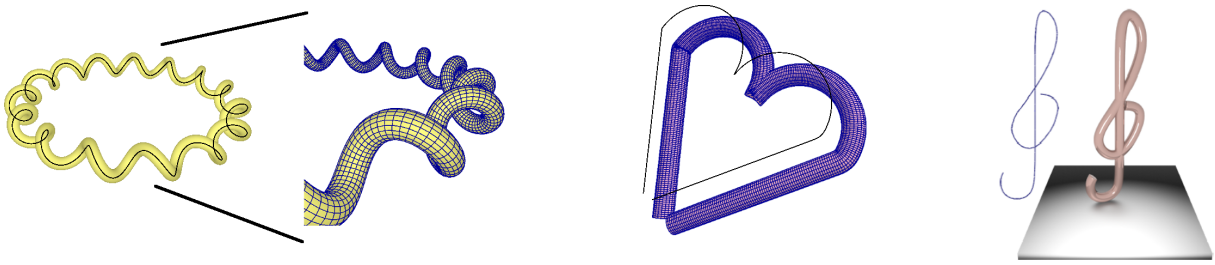


FIGURE 5 – Exemples de cylindres généralisés pour différentes courbes d'extrusions.

Question 5 *Soit $C : t \to C(t)$ la courbe d'extrusion à suivre. Donnez l'equation cartésienne de la surface tubulaire S associée.*

Question 6 *Affichez un cercle unitaire centré autour de 0 et normale à $u_z = (0, 0, 1)$.*
Affichez un cercle de rayon $r = 0.2$ centré autour de $x_0 = (0.1, 0.1, 0.2)$ et normale à $n = (1, 1, 1)/\sqrt{3}$.

Question 7 *Soit y_0 et y_1 deux points de l'espace. Construisez et affichez le cylindre de rayon r associé au segment reliant y_0 et y_1 (voir fig. 6).*

Question 8 *Soit une coube quelconque de l'espace formée par un ensemble de segments (=polyline).*
Construisez le cylindre généralisé associé à cette courbe.
Commentez et expliquez vos résultats (conditions de non inter-collision, torsion, ...).

Question 9 (extra) *Généralisez pour le cas d'extrusion non cylindrique : On désigne ce type de surface par* sweep surfaces *(voir fig. 6).*

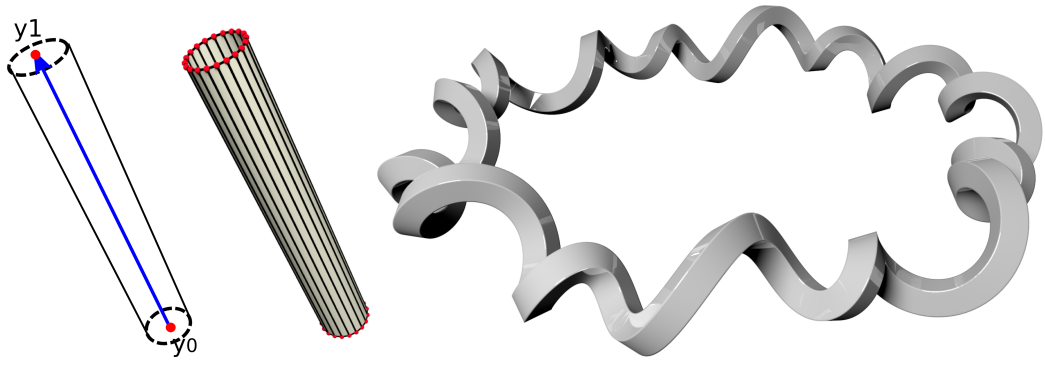


FIGURE 6 – Gauche : Cylindre simple reliant les points y_0 et y_1. Droite : *Sweep surface* obtenue par extrusion d'un carré le long d'une courbe 3D.