



R my Malgouyres,  
Algorithmes pour la synth se d'images  
et l'animation 3D,  
DUNOD, 2002

## TP n  7 : Construction de poly dres

### Objectifs :

Le but de ce TP est de construire des poly dres correspondant   :

1. Une bo te (c'est   dire un parall lepip de rectangle) ;
2. Un c ne de r volution ;
3. Une sph re.
4. Des surfaces cylindriques et de r volution   partir de courbes splines.

On impl mentera ces constructions dans les constructeurs appropri s de classes donn es ci-dessous dans une application existante dont l'interface est donn e. Cette application comprend d j  une fonction d'exportation de poly dres qui permettra de visualiser les r sultats.

## 1 Les classes sous-jacentes

On t l chargera et on d compressera le fichier *ProjetVisual3D.zip*, qui contient un projet *Visual C++* (r pertoire *TP3D*), et un ex cutable *French3D.exe* qui permet de tester et comparer les r sultats. Le projet *Visual C++* fourni comprend d j  une interface et quelques fonctionnalit s qui permettent, en impl mentant les fonctions essentielles, d'arriver rapidement   un r sultat int ressant, et de tester en profondeur les algorithmes.

### 1.1 La classe Point3D

La classe `Point3D` est destin e   repr senter   la fois les points de  $\mathbb{R}^3$  et les vecteurs de  $\mathbb{R}^3$ . Cette classe se trouve dans le fichier *Point3D.h* et l'impl mentation des fonctions se trouve dans le fichier *Objet3D.cpp*. Toutes les fonctions de la classe `Point3D` sont d j  impl ment es ; on ne doit donc pas les r  crire, mais on pourra les utiliser librement.

```
class Point3D{
public:
    double x, y, z;    // coordonn es

    // Constructeurs :
    Point3D(double a, double b, double c){x = a; y = b; z = c;}
    Point3D(void){x = y = z = 0.0;}
```

```

    // Operations de base~:
    Point3D operator+(const Point3D&); // addition
    Point3D operator-(const Point3D&); // soustraction
    Point3D operator-(void); // oppose
    double operator*(const Point3D&); // produit scalaire
    Point3D operator^(const Point3D&); // produit vectoriel

    // multiplication par un nombre (à droite et à gauche)
    friend Point3D operator*(double, const Point3D&);
    friend Point3D operator*(const Point3D&, double);

    // divise un vecteur par sa norme, retourne false si la norme est nulle :
    bool normer(void);

    // retourne la norme d'un vecteur (ou la distance d'un point a l'origine 0)
    double norme(void);
};

```

## 1.2 La classe Sommet

La classe `sommet` représente les sommets des polyèdres. Chaque sommet possède une position dans l'espace, qui est un point de  $\mathbb{R}^3$ , un vecteur normal, et d'autres données que nous n'utiliserons que plus tard.

```

class Sommet{
public:
    Point3D pos; // Position dans l'espace
    Point3D normale; // vecteur normal
    Intensitesgouraud intens; // intensités pour éclairnement de Gouraud
    Point3D dpsds; // dérivée partielle pour bumpmap
    Point3D dpsdt; // dérivée partielle pour bumpmap
    double coordtextureX, coordtextureY; // coordonnée de texture

    Sommet operator+(const Sommet&); // addition
    Sommet operator-(const Sommet&) const; // soustraction
    friend Sommet operator*(double, const Sommet&); // multiplication par un nombre
    friend Sommet operator*(const Sommet&, double); // multiplication par un nombre
    Sommet operator+(const Point3D& p); // addition
};

```

Pour le moment, lors de la construction des polyèdres, nous initialiserons simplement la position et le vecteur normal de chaque sommet.

## 1.3 La classe Facette

La classe `Facette` (qui se trouve dans le fichier `Objet3D.h`) permet de représenter les facettes d'un polyèdre. Une facette étant une suite de numéros de sommets, la classe `facette` contient un entier `nbsomm` représentant le nombre de sommets de la facette, et un tableau d'entier permettant de stocker les numéros de sommets de la facette.

```

class Facette{

    int nbsomm;    // nombre de sommets de la facette
    int * tabnosomm;    // tableau des numeros de sommets de la facette
                    // indexees sur un tableau de sommet
                    // défini dans la classe Objet3D

public:

    // constructeurs, destructeur, operateur =
    Facette(void){nbsomm = 0; tabnosomm = NULL;}
    Facette(int nb);
    ~Facette(){delete [] tabnosomm; tabnosomm = NULL; nbsomm = 0;}
    Facette(Facette &);
    Facette & operator=(Facette &);
};

```

Toutes les fonctions de la classe `Facette` sont déjà implémentées (dans le fichier `Objet3D.cpp`); on ne doit donc pas les réécrire.

## 1.4 La classe `Objet3D`

La classe `Objet3D` est une classe générique qui sera la base de tous les objets 3D considérés au cours des TP. Cette classe contient, outre des données sur la couleur de l'objet, un polyèdre. En effet, tous les objets considérés seront facettisés. Le but du TP est de construire ce polyèdre pour des boîtes, cônes sphères,...

Le polyèdre est représenté comme suit :

- Un entier `nbsomm` représente le nombre de sommets du polyèdre ;
- Un tableau de `Sommet` nommé `tabsomm` représente les sommets du polyèdre ;
- Un entier `nbfaces` représente le nombre de faces du polyèdre ;
- Un tableau de `Facette` représente les faces du polyèdre.

La classe `Objet3D` se trouve dans le fichier `Objet3D.h`.

```

class Objet3D{

friend class Scene3D;

protected:
    Sommet *tabsomm;    // tableau des sommets du polyedre
    int nbsomm;        // nombre de sommets du polyedre
    int nbfaces;      // nombre de faces du polyedre
    Facette* faces;   // tableau des faces du polyedre

    Matrice rotation; // matrice de passage du repere fixe de la scene au repere lie a l'o

    Point3D origine;  // translation du repere de l'objet par rapport au repere de la scen
    double scalex, scaley, scalez; // changements d'echelle sur l'objet
    double rotx, roty, rotz;

```

```

    // donn es n cessaires aux mod les d'illumination :

MaterialData material;
TextureData textures;

public:
    // les fonctions suivantes sont d ja impl ment es

    // constructeurs et destructeur, op rateur =
Objet3D();
virtual ~Objet3D();
Objet3D(Objet3D &);
Objet3D(double x, double y, double z,
        double scax, double scay, double scaz,
        double rx, double ry, double rz,
        MaterialData mat, const TextureData &text);
Objet3D & operator=(Objet3D &);

    // recopies d'un objet 3D :

    // virtuel, permet de recopier les donn es propres aux objets d riv s :
virtual Objet3D * copie(void)=0;
void copieObjet3D(Objet3D&);
};

```

L  non plus, on ne doit pas impl menter les fonctions apparaissant dans cette classe, qui sont d j   crites dans le fichier *Objet3D.cpp*.

Le constructeur d'Objet3D ayant 11 param tres sera appel  par chaque constructeur pour les sph res, c nes,... Cet appel est d j   crit.

## 2 Travail   r aliser

### 2.1 Constructeur de boite

Une boite est un parall l pip de rectangle, surface de l'ensemble des points  $(x, y, z)$  tels que;

$$x_{min} \leq x \leq x_{max} \text{ et } y_{min} \leq y \leq y_{max} \text{ et } z_{min} \leq z \leq z_{max}$$

o   $x_{max}, y_{max}, z_{max}, x_{min}, y_{min}, z_{min}$  sont des nombres r els.

La classe Boite contient les donn es  $x_{max}, y_{max}, z_{max}, x_{min}, y_{min}, z_{min}$ . La classe Boite h rite de la classe Objet3D, ce qui permet de repr senter le poly dre sous-jacent   la boite. La classe Boite se trouve dans le fichier *Objetderiv.h*.

```

class Boite : public Objet3D{

    double xmax, ymax, zmax, xmin, ymin, zmin;

```

```

public:
    Boite(void):Objet3D();
    Boite(double xmi, double xma, double ymi, double yma, double zmi, double zma);
    // constructeur à implémenter :
    Boite(double x1, double y1, double z1, double dx, double dy, double dz,
           int nbmer,
           double rotx, double roty, double rotz,
           double scalex, double scaley, double scalez,
           MaterialData mat, const TextureData &text);

    Objet3D * copie(void);
};

```

Le premier des buts du TP est d'implémenter le corps du troisième constructeur de la classe `Boite`, qui se trouve dans le fichier `Objetderiv.cpp`. Seuls les 6 premiers paramètres seront pour le moment pris en compte. les paramètres `x1, y1, z1` donnant le centre de la boîte, et les paramètres `dx, dy, dz` donnant les dimensions de la boîte suivant les 3 axes. Ceci permet d'initialiser les données `xmin, xmax ymin, ymax zmin` et `zmax` de la boîte.

On doit construire les sommets et facettes de la boîte en allouant et initialisant les données `nbsomm, tabsomm, nbfaces` et `faces` de la classe de base `Objet3D`. Pour chaque sommet, on doit déterminer la position et la normale (qui doit être de norme 1) au sommet (voir la classe `Sommet`). Pour chaque facette, on doit déterminer le nombre de sommets de la facette et allouer et initialiser le tableau des numéros de sommets (voir la classe `Facette`).

## 2.2 Tester

Pour tester, On exécutera le programme, et on activera les commandes de menu suivantes :

1. *File/New Scene* ; (crée une nouvelle scène)
2. *Insert/Box* ; (insère une boîte dans la scène)
3. *File/Export/Polytext*. (enregistre le polyèdre créé dans un fichier)

Le constructeur de boîte est alors appelé automatiquement, et un fichier ".plt" est créé.

Pour visualiser le résultat, lancer l'exécutable *French 3D* (fourni), et exécuter les commandes de menu suivante :

1. *File/New Scene* ; (crée une nouvelle scène)
2. *File/Import/Polytext* ; (charger en mémoire le polyèdre du fichier)
3. *Insert/Light Source*. (insérer une source lumineuse).

Une boîte doit alors apparaître. L'affichage doit fonctionner avec ou sans lissage, avec le lissage de Phong et avec le lissage de Gouraud (pour activer ou désactiver le lissage, régler dans *Display/Options éclairément*. Si l'affichage fonctionne sans lissage mais ne fonctionne pas avec le lissage de Phong ou de Gouraud, le vecteur normal aux sommets est probablement en cause.

Pour visualiser la boîte sous différents angles, on pourra exécuter la commande *Display/Contrôles caméra* qui ouvre les contrôles de navigation. On pourra aussi modifier la position de la source lumineuse (*Edit/Light Source*), ou insérer d'autres sources lumineuses.

## 2.3 Constructeur de cône de révolution

La classe `Conerevoltronq` permet de représenter un cône de révolution. Comme la classe `Boite`, la classe `Conerevoltronq` hérite de la classe `Objet3D`, ce qui permet de représenter le polyèdre résultant de la facettisation. La classe `Conerevoltronq` se trouve dans le fichier *Objetderiv.h*.

```
class Conerevoltronq : public Objet3D{

    double rayon, hauteur, rayontronq;
    int nbmeridien, nbparallele;

public:
    Conerevoltronq(void):Objet3D(){rayon = hauteur = 0; nbmeridien = 0;}
    Conerevoltronq(double r, double h, double rt, int m, int p):
        Objet3D(){rayon=r; hauteur=h; rayontronq = rt; nbmeridien=m; nbparallele=p;}
    Conerevoltronq(double xor, double yor, double zor,
        double rotx, double roty, double rotz,
        double scalex, double scaley, double scalez,
        double r, double hauttronq, double rt,
        int nbmerid, int nbparal,
        MaterialData mat, const TextureData &text);

    Objet3D * copie(void);
};
```

Le deuxième des buts du TP est d'implémenter le corps du troisième constructeur de la classe `Conerevoltronq`, qui se trouve dans le fichier *Objetderiv.cpp*. Pour le moment, seuls les paramètres `r`, `hauttronq`, `rt`, `nbmerid`, représentant le rayon, la hauteur, la hauteur, le rayon de troquature et le nombre de méridiens doivent être pris en compte. Ces paramètres permettent d'initialiser les données membre `rayon`, `hauteur`, `rayontronq`, `nbmeridien`.

Comme dans le cas de la boîte, il faut construire le polyèdre correspondant au cône de révolution par facettisation, puis tester comme pour la boîte.

L'affichage doit fonctionner avec ou sans lissage, avec le lissage de Phong et avec le lissage de Gouraud (pour activer ou désactiver le lissage, régler dans *Display/Options éclairément*). Si l'affichage fonctionne sans lissage mais ne fonctionne pas avec le lissage de Phong ou de Gouraud, le vecteur normal aux sommets est probablement en cause.

## 2.4 Constructeur de sphère

La classe `Sphere` permet de représenter une sphère. Comme la classe `Boite` et la classe `Conerevoltronq`, la classe `Sphere` hérite de la classe `Objet3D`, ce qui permet de représenter le polyèdre résultant de la facettisation. La classe `Sphere` se trouve dans le fichier *Objetderiv.h*

```
class Sphere : public Objet3D{

    double rayon;
    int nbmeridien, nbparallele;
```

```

public:
    Sphere(void):Objet3D(){rayon = 0; nbparallele = nbmeridien = 0;}
    Sphere(double r, int m, int p):Objet3D(){rayon=r; nbparallele=p; nbmeridien=m;}
    Sphere(double xc, double yc, double zc,
            double rotx, double roty, double rotz,
            double scalex, double scaley, double scalez,
            double r,
            int nbparal, int nbmerid,
            MaterialData mat, const TextureData &text);

    Objet3D * copie(void);
};

```

Le troisième des buts du TP est d'implémenter le corps du troisième constructeur de la classe `Sphere`, qui se trouve dans le fichier `Objetderiv.cpp`. Pour le moment, seuls les paramètres `r`, `nbparal`, `nbmerid`, représentant le rayon, le nombre de parallèles et le nombre de méridiens doivent être pris en compte.

Comme dans le cas de la boîte ou du cône de révolution, il faut construire le polyèdre correspondant à la sphère par facettisation, puis tester comme pour la boîte. L'affichage doit fonctionner avec ou sans lissage, avec le lissage de Phong et avec le lissage de Gouraud (pour activer ou désactiver le lissage, régler dans *Display/Options éclairement*. Si l'affichage fonctionne sans lissage mais ne fonctionne pas avec le lissage de Phong ou de Gouraud, le vecteur normal aux sommets est probablement en cause.

### 3 Extrusion simple

Voici la classe `Cylinspline`, représentant les durfaces définies par extrusion simple, définie dans le fichier `Objetderiv.h` et dont l'implémentation des fonction se trouve dans le fichier `ObjetSpline.cpp`.

```

class Cylinspline : public Objet3D{
    double hauteur; // hauteur de la surface extrusion simple
    int echantillonage;
    int nbparal; // nombre de parallèles
    Spline2D * spline2d; // pointeur sur la courbe 2D
    double topreduction; // rapport rho
public:
    Cylinspline(void);
    Cylinspline(double h, int ech, Spline2D * sp);
    Cylinspline(double xor, double yor, double zor,
            double rotx, double roty, double rotz,
            double scalex, double scaley, double scalez,
            Spline2D * sp, double h,
            double factor_reduc, // rapport rho
            int ech, // nombre de points calculés pour chaque partie de la spline 2D
            int nbparallele,
            MaterialData mat, const TextureData &text);
    Objet3D * copie(void);
};

```

```
};
```

Les données membres sont les suivantes :

1. `hauteur` représente la hauteur de la surface cylindrique ;
2. `topreduction` qui représente le rapport  $\rho$  de l'extrusion simple ;
3. `echantillonnage` représente le nombre de points pris sur chaque portion de la courbe spline. Le nombre de méridiens sera égal à  $(\text{nbctrlpoints}-3) * \text{echantillonnage}$ , où `nbctrlpoints` est le nombre de points de contrôle de la courbe spline.
4. `spline2d` pointe sur une `Splinefermee` (classe dérivée de la classe `Spline2D` comme on l'a vu dans les TP sur les courbes  $B$ -splines. C'est cette courbe spline qui constituera la base de la surface cylindrique à construire. La classe `Spline2D` représentant les courbes  $B$ -splines cubiques est déjà implémentée. Pour calculer des points de la courbe, on utilisera la fonction membre
 

```
void getpoint(double t, double & xpos, double & ypos) ;
```

 qui donne la position  $(xpos, ypos)$  d'un point de la courbe en fonction du paramètre  $t \in [0, 1]$ .
5. `nbparal` qui représente le nombre de parallèles pour la facettisation.

Comme on peut le constater, la classe `Cylinspline` hérite de la classe `Objet3D`. Comme dans le cas des quadriques ci-dessus, le travail à réaliser consiste à construire le polyèdre représenté par les données membres `tabsomm` et `faces` de la classe `Objet3D`. Ce travail sera fait dans le constructeur de `Cylinspline` ayant 16 paramètres et dont l'implémentation à compléter se trouve dans le fichier `ObjetSpline.cpp`. Pour le moment, seuls les paramètres `sp`, `ech`, `h` et `nbparallele` correspondant aux données membres de la classe `Cylinspline` seront utilisés.

Pour tester, exécuter le programme, et activer le menu *Splines/vue spline 2D*, une nouvelle fenêtre apparaît. Exécuter le menu *Splines/Nouvelle spline fermée*, et dessiner une courbe spline. Après le click droit, entrer un nom pour la courbe spline, il vous sera nécessaire pour insérer la spline extrusion simple (menu *Insert/Spline 3D/Simple extrusion*). Exporter dans un fichier *Polytext*, et chargez le fichier dans l'exécutable fourni pour visualiser.

## 4 Surfaces de révolution sphériques

Un travail analogue à celui réalisé pour les surfaces cylindriques doit être fait pour les surfaces de révolution sphériques. Pour cela, on travaillera sur la classe `Spline3Drevol` ci-dessous, qui se trouve aussi dans le fichier `Objetderiv.h`.

```
class Spline3Drevol : public Objet3D{
    int echantillonnage;
    int nbmeridiens;
    Spline2D * spline2d;
public:
    Spline3Drevol(void);
    Spline3Drevol(int ech, int nbm, Spline2D * sp);
    Spline3Drevol(double xor, double yor, double zor,
                  double rotx, double roty, double rotz,
                  double scalex, double scaley, double scalez,
```

```

        Spline2D * sp,
        int nbmerid, int ech,
        MaterialData mat, const TextureData &text);
Objet3D * copie(void);
};

```

Pour tester, ex cuter le programme, et activer le menu *Splines/vue spline 2D*, une nouvelle fen tre appara t. Ex cuter le menu *Splines/Nouvelle spline pour surface de r volution*, et dessiner une courbe spline. Apr s le click droit, entrer un nom pour la courbe spline, il vous sera n cessaire pour ins rer la surface spline de r volution (menu *Insert/Spline 3D/Revolution surface*). Exporter dans un fichier *Polytext*, et chargez le fichier dans l'ex cutable fourni pour visualiser.

## 5 Rotation, translation, changement d' chelle

Pour le moment, les objets construits sont toujours centr s sur l'origine et en position verticale. Pour pouvoir translater et tourner ces objets, il faut mettre le code suivant   la suite de la construction des sommets dans chaque constructeur :

```

    // changements d'echelle suivant les differents axes,
    // rotations, et translation
for (i = 0 ; i < nbsomm ; i++){
    tabsomm[i].pos.x *= scalex;
    tabsomm[i].pos.y *= scaley;
    tabsomm[i].pos.z *= scalez;
    tabsomm[i].dpsds.x *= scalex;
    tabsomm[i].dpsds.y *= scaley;
    tabsomm[i].dpsds.z *= scalez;
    tabsomm[i].dpsdt.x *= scalex;
    tabsomm[i].dpsdt.y *= scaley;
    tabsomm[i].dpsdt.z *= scalez;
    tabsomm[i].normale.x /= scalex;
    tabsomm[i].normale.y /= scaley;
    tabsomm[i].normale.z /= scalez;
    tabsomm[i].normale.normer();
    tabsomm[i] = rotation*tabsomm[i];
    tabsomm[i].pos = tabsomm[i].pos + origine;
}

```

On pourra copier ce code qui se trouve dans le constructeur d j   impl ment  de la classe *Cylindre*.