

---

## TP1 - Introduction à la modélisation 3D

---

### 1 Création d'un maillage triangulaire de sphère 3D

Dans le fichier `tp.cpp`, compléter la fonction `void setUnitSphere(Mesh &o_mesh, int nX, int nY)`, qui créera un maillage triangulaire de sphère 3D.

1. Créer des sommets discrétisant la sphère avec un nombre  $nX$  de méridiens et  $nY$  de parallèles.  
Indications : un point 3D sur la sphère peut être obtenu à l'aide de la paramétrisation sphérique en fonction de deux angles  $(\theta, \varphi) \in [0, 2\pi] \times [-\pi/2, \pi/2]$  :
  - $x = \cos(\theta) * \cos(\varphi)$
  - $y = \sin(\theta) * \cos(\varphi)$
  - $z = \sin(\varphi)$
2. Générer la topologie : créer la liste des triangles du maillage.
3. Ajouter la fonctionnalité suivante : l'appui de la touche « + » augmente le nombre de méridiens et de parallèles de 1. De la même manière, l'appui sur la touche « - » diminue de 1 le nombre de méridiens et de parallèles.
4. Ajouter les normales aux sommets.

1. Pour créer les sommets, j'ai complété la fonction `setUnitSphere` :

```
void setUnitSphere( Mesh & o_mesh, int nX, int nY)
{
    o_mesh.vertices.clear();
    o_mesh.normals.clear();
    o_mesh.triangles.clear();
    float x, y, z, teta, phi = 0;
    for (int i = 0; i < nX; i++) {
        for (int j = 0; j < nY; j++) {
            teta = i*(2*M_PI)/(nX-1);
            phi = (j*M_PI/(nY-1))-(M_PI/2);
            x = cos(teta)*cos(phi);
            y = sin(teta)*cos(phi);
            z = sin(phi);
            Vec3 pts = Vec3(x, y, z);
            o_mesh.vertices.push_back(pts);
        }
    }
}
```

2. Pour générer ma topologie, j'ai simplement vu ma sphère comme une grille : chaque parallèle correspond à un point sur l'axe des ordonnées, et chaque méridien à un point sur l'axe des abscisses :

```
void setUnitSphere( Mesh & o_mesh, int nX, int nY)
{
    o_mesh.vertices.clear();
    o_mesh.normals.clear();
```

```

o_mesh.triangles.clear();
float x, y, z, teta, phi =0;
for (int i = 0; i < nX; i++) {
    for (int j = 0; j < nY; j++) {
        teta = i*(2*M_PI)/(nX-1);
        phi = (j*M_PI/(nY-1))-(M_PI/2);
        x = cos(teta)*cos(phi);
        y = sin(teta)*cos(phi);
        z = sin(phi);
        Vec3 pts = Vec3(x, y, z);
        o_mesh.vertices.push_back(pts);
    }
}
for (int i = 0; i < nX; i++) {
    for (int j = 0; j < nY; j++) {
        int pt1 = i*nY+j;
        if (i!=0 && j!=0) {
            int pt2 = (i-1)*nY+j;
            int pt3 = i*nY+(j-1);
            o_mesh.triangles.push_back(Triangle(pt1,pt2,pt3));
        }

        if (i!=(nX-1) && j!=(nY-1)) {
            int pt4 = (i+1)*nY+j;
            int pt5 = i*nY+(j+1);
            o_mesh.triangles.push_back(Triangle(pt1,pt4,pt5));
        }
    }
}
}
}

```

3. Ici, openGL travaille pour nous : il suffit de modifier la fonction `key` pour ajouter les cas « + » et « - ». Nous devons cependant instancier deux variables globales pour garder le nombre de parallèles et méridiens à chaque itération. Il ne faut pas oublier de remettre à zéro les vecteurs de notre mesh, pour ne pas créer des choses bizarres :

```

// Variables globales pour les parallèles et méridiens
int nX = 20;
int nY = 20;

// Modification de la fonction key :
void key (unsigned char keyPressed, int x, int y) {
    switch (keyPressed) {
        // ...
        case '+':
            nX++;
            nY++;
            setUnitSphere(unit_sphere, nX, nY);
            break;

        case '-':

```

```

        nX--;
        nY--;
        setUnitSphere(unit_sphere, nX, nY);
        break;
    // ...
}
}

```

4. Ici, on peut tricher. En effet, le cercle est de rayon 1, et les normales sont les vecteurs du centre  $O = (0, 0)$  et de chaque point, les normales sont donc directement les coordonnées des points :

```

// Notre fonction :
void setUnitSphere(Mesh &o_mesh, size_t nX, size_t nY) {
    o_mesh.normals.push_back(pts/sqrt(x*x + y*y + z*z));
    // ...
}

```