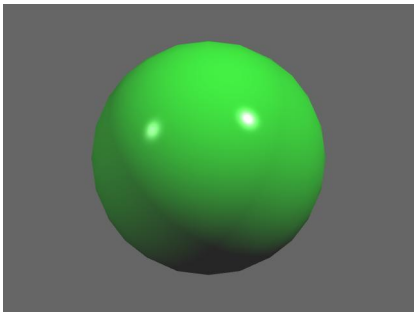
Sphères

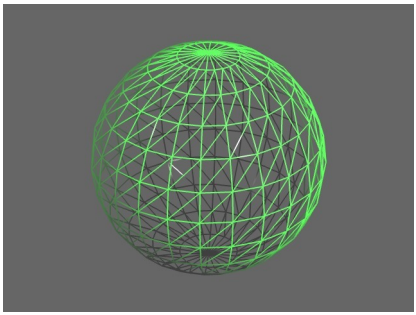
Voyons maintenant ce qu'il en est avec les sphères, par exemple celle de Figure 16.20. J'ai fait exprès de rendre les facettes assez visibles sur les bords en ne créant que peu de sommets pour les définir.

Figure 16.20 : *Sphère éclairée*



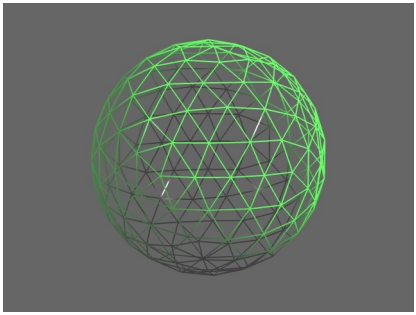
Une sphère comme celle-ci peut être construite de deux manières. La première est basée sur un système de coordonnées longitude et latitude comme sur le globe terrestre. Cela donne la Figure 16.21 où j'ai dessiné seulement les contours des triangles.

Figure 16.21 : *Sphère longitude-latitude*



La seconde méthode procède par subdivision. Elle produit une sphère avec une structure différente qui ressemble plutôt à la Géode de Paris ou à l'une des salles du Futuroscope à Poitiers. Une telle sphère est appelée *icosphère* dans Blender.

Figure 16.22 : *Sphère par subdivision*



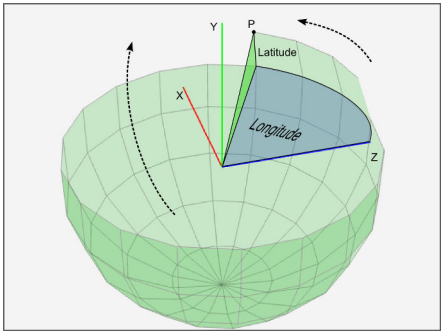
Dans les deux cas, ce sont des sphères de rayon 1 centrées en (0,0,0). Il est simple d'appliquer une transformation matricielle pour construire toute autre sphère. L'intérêt de cette sphère est que les normales ont les mêmes coordonnées que les sommets.

Je vais maintenant vous expliquer comment ces deux structures sont calculées.

Génération par longitude-latitude

La méthode ressemble à celle employée jusque-là : cône et cylindre. Il s'agit d'une nappe rectangulaire repliée sur elle-même en longitude et déformée pour arriver à une sphère. La Figure 16.23 montre la sphère en cours de création. J'ai dessiné des quads pour ne pas compliquer la vue.

Figure 16.23 : *Sphère en cours de création*

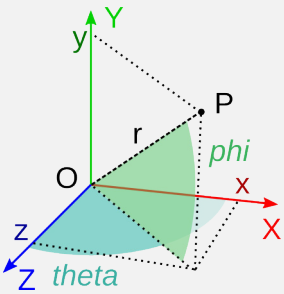


Je définis un système de coordonnées comme à la surface de la Terre : une *longitude* et une *latitude*. La longitude est comptée de 0 à 360°, à partir de l'axe Z, en allant vers l'axe X (c'est l'inverse sur les cartes de la Terre). La latitude est comptée de −90° au niveau du pôle sud et +90° au niveau du pôle nord. Vous l'aurez deviné, je vais définir les points selon ces deux angles et en convertissant les coordonnées (*longitude*, *latitude*) en coordonnées cartésiennes. Ces coordonnées sont dites sphériques.

Coordonnées sphériques

Voici un rappel de ce que sont les coordonnées sphériques. Ce sont deux angles, *theta* et *phi* et une distance *r*. Par contre, je n'ai pas pris la même définition que celle de Wikipédia pour me faciliter les calculs et pour imiter le système de coordonnées terrestre (voir Figure 16.24).

Figure 16.24 : *Coordonnées sphériques*



L'angle *theta* est compté à partir de Z, allant vers X. C'est l'angle d'azimut du chapitre Construction d'une scène complexe et en géographie, on l'appelle angle de *longitude*. L'angle *phi* est compté à partir du plan OZX. On l'appelle *latitude* en géographie. C'est l'angle de *hauteur* ou *élévation* du chapitre Construction d'une scène complexe. Voici les formules qui permettent de passer des coordonnées sphériques aux coordonnées cartésiennes.

$$\begin{cases} x = r * \cos(phi) * \sin(theta) \\ y = r * \sin(phi) \\ z = r * \cos(phi) * \cos(theta) \end{cases}$$

Les deux angles *theta* et *phi* sont fournis par le parcours des sommets de la nappe. Il faut deux boucles imbriquées pour arpenter la nappe entière. Voici la mise en œuvre.

Exemple 16.10 : *Création d'une sphère*

```
vec3 SphereLonLat::getCoord(float a, float b)
{
    // longitude en radians
    float lon = a * 2.0 * Math.PI;

    // latitude en radians
    float lat = (b - 0.5) * PI;

    // conversion sphériques -> cartésiennes
    float x = cos(lat) * sin(lon);
    float y = sin(lat);
    float z = cos(lat) * cos(lon);
    return vec3(x,y,z);
}

SphereLonLat::SphereLonLat(int nbLon, int nbLat)
{
    // créer le maillage : une nappe rectangulaire
    Mesh mesh = new Mesh("sphere");
    MeshModuleTopology topology = new MeshModuleTopology(mesh);
    int num0 = topology.addRectangularSurface(
            nbLon, nbLat, "sphere %d-%d", true, false);

    // aller du pôle sud au pôle nord, pour définir les sommets
    for (int ilat=0; ilat<nbLat; ilat++) {
        // faire le tour du globe
        for (int ilon=0; ilon<nbLon; ilon++) {
            // récupérer le sommet
            int num = ilon + ilat*nbLon + num0;
            Vertex vertex = mesh.getVertex(num);
            // calculer ses coordonnées
            float a = (float)ilon / nbLon;          // ATTENTION !
            float b = (float)ilat / (nbLat-1);      // ATTENTION !
            vec3 coords = getCoord(a, b);
            vertex.setCoord(coords);
            vertex.setNormal(coords);
        }
    }
}
```

Comme d'habitude, il y a le piège à loups de la division entre entiers selon le langage de programmation, au niveau du calcul de `a` et `b` en fonction des variables de la boucle.

Note › *Les sources complètes sont dans le dossier* `SphereLonLat`.

Vous aurez sûrement remarqué que ma méthode crée une accumulation de points au niveau des pôles, exactement comme pour le cône. Il aurait été plus économique de créer

deux nappes circulaires centrées sur les pôles, recourbées vers l'équateur, et d'ajouter des quadrilatères pour couvrir l'équateur. Ma motivation est de pouvoir plaquer une image appelée texture sur la sphère. La génération des coordonnées de texture correspondant à une image n'est facile qu'avec une nappe rectangulaire ou hexagonale. À ce sujet, pour que les coordonnées de texture soient correctes, il ne faut surtout pas replier la nappe en longitude, mais au contraire avoir une colonne de trop, la dernière ayant les mêmes coordonnées 3D que la première. L'Exemple 16.11 donne une modification de l'algorithme pour définir les coordonnées de texture correctement. Remarquez que le calcul de `a` est le seul endroit où on n'emploie pas `nbLCon+1`. C'est ce qui fait que les derniers sommets en longitude auront les mêmes coordonnées que les premiers.

Exemple 16.11 : *Création d'une sphère avec coordonnées de texture*

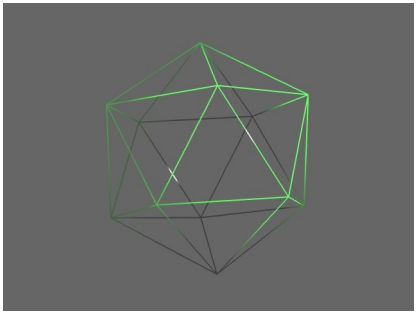
```
SphereLonLat::SphereLonLat(int nbLon, int nbLat)
{
    // créer le maillage : une nappe rectangulaire
    Mesh mesh = new Mesh("sphere");
    MeshModuleTopology topology = new MeshModuleTopology(mesh);
    int num0 = topology.addRectangularSurface(
            nbLon+1, nbLat, "sphere %d-%d", true, false);

    // aller du pôle sud au pôle nord, pour définir les sommets
    for (int ilat=0; ilat<nbLat; ilat++) {
        // faire le tour du globe
        for (int ilon=0; ilon<=nbLon; ilon++) {
            // récupérer le sommet
            int num = ilon + ilat*(nbLon+1) + num0;
            Vertex vertex = mesh.getVertex(num);
            // calculer ses coordonnées
            float a = (float)ilon / nbLon;          // ATTENTION !
            float b = (float)ilat / (nbLat-1);      // ATTENTION !
            vec3 coords = getCoord(a, b);
            vertex.setCoord(coords);
            vertex.setNormal(coords);
            vertex.setTexCoord(vec2.fromValues(a,b));
        }
    }
}
```

Génération par subdivision

Cette méthode pour générer une sphère est entièrement différente. Elle consiste à partir d'une forme assez simpliste et à la subdiviser, c'est-à-dire qu'on remplace tous ses triangles par de plus petits. Il faut partir d'une forme à peu près sphérique, sinon le résultat n'est pas satisfaisant (voir Figure 16.25).

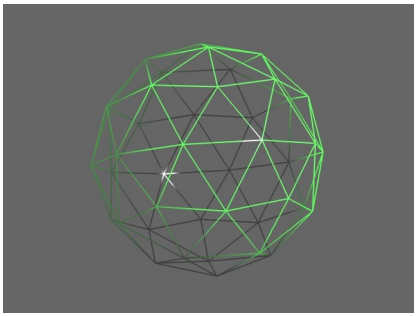
Figure 16.25 : *Objet de départ*



Cette forme s'appelle un icosaèdre. Il contient 20 triangles car εἴκοσι (ikosi) en grec signifie vingt. Pour en savoir davantage, je vous renvoie à la page de Wikipédia. On peut aussi partir d'un tétraèdre, d'un cube ou d'un octaèdre, mais ils sont davantage patatoïdes et donnent des sphères moins régulières à nombre égal de triangles.

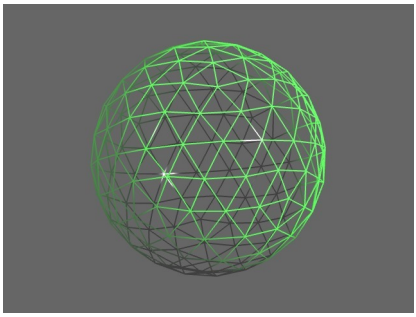
Puis je découpe ces 20 triangles pour en créer 80 (voir Figure 16.26).

Figure 16.26 : *Étape 1*



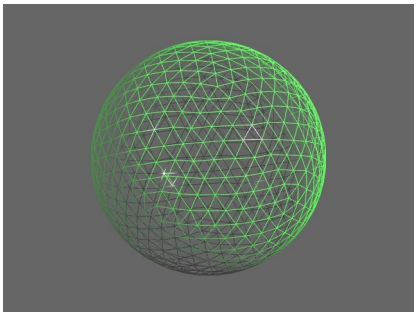
Avez-vous remarqué que le triangle de devant était maintenant un peu plié ? Il est remplacé par quatre triangles qui ne sont plus coplanaires. En fait, chaque fois que je découpe un triangle, je repousse les milieux afin de donner une courbure au résultat.

Figure 16.27 : *Étape 2*

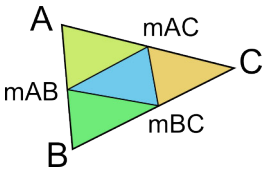


La forme est déjà de bonne qualité. C'est celle qui a servi à faire l'illustration initiale de cette section. Je peux faire une itération de plus pour obtenir la Figure 16.28.

Figure 16.28 : *Étape 3*



Ce n'est pas facile à voir car les triangles sont assez petits, mais d'une étape à l'autre, j'ai découpé chaque triangle en quatre plus petits, en m'appuyant sur les milieux des côtés. J'ai fait appel pour cela à la méthode `subdiviserTous` de la classe Maillage (pour plus d'informations sur cette méthode se reporter à la section Subdivision).



La nouveauté par rapport à la subdivision, c'est que les milieux ne restent pas "à leur place" : je les repousse vers l'extérieur afin que leur distance au centre de la sphère

devienne exactement 1. Pour cela, après subdivision, je passe tous les sommets en revue et je normalise leurs coordonnées. J'en profite pour affecter le vecteur normal avec les mêmes coordonnées puisqu'il s'agit d'une sphère unité.

Exemple 16.12 : *Normalisation des coordonnées*

```
for (Vertex v: m_Mesh.getVertices()) {
    vec3 coords = v.getCoord();
    vec3.normalize(coords, coords);
    v.setCoord(coords);
    v.setNormal(coords);
}
```

La construction de la forme de départ est assez subtile. Il y a d'une part les coordonnées des sommets et d'autre part les triangles à bâtir. Wikipédia montre comment on calcule les coordonnées avec le nombre d'or. Par exemple, l'un des points se trouve en $(0, (1+\sqrt{5})/2, 1)$. Ces coordonnées sont à modifier pour rendre leur norme égale à 1, cela donne par exemple $(0, k1, k2)$ avec $k1{=}0.85$ et $k2{=}0.52$. Voici un extrait de l'algorithme qui crée l'icosaèdre, en JavaScript :

Exemple 16.13 : *Création d'un icosaèdre*

```
float k2 = 0.52573;
float k1 = 0.85065;

// définition des sommets
Vertex A = mesh.addVertex("A");
A.setCoord(vec3.fromValues( k2, 0,-k1));
Vertex B = mesh.addVertex("B");
B.setCoord(vec3.fromValues(-k2, 0,-k1));
Vertex C = mesh.addVertex("C");
C.setCoord(vec3.fromValues( k2, 0, k1));
...

// ajout des triangles de cette forme
mesh.addTriangle(A,E,B);
mesh.addTriangle(A,J,E);
mesh.addTriangle(J,F,E);
mesh.addTriangle(E,F,I);
...
```

Note *› Les sources complètes sont dans le dossier `SphereTess` avec les variantes pour chaque plate-forme. Je vous invite à aller les regarder. Mais attention, les programmes sont légèrement plus complexes que dans ce livre à cause des particularités des structures de données de chaque langage.*

Attention *› Vous ne pourrez pas employer cette méthode de création de sphère pour plaquer une texture. Il y a un problème au niveau des bords qui empêche de définir des coordonnées de texture satisfaisantes.*