



ENSEIGNEMENT DE PROMOTION ET DE FORMATION CONTINUE DE L'UNIVERSITE  
LIBRE DE BRUXELLES ET DE LA CHAMBRE DE COMMERCE ET D'INDUSTRIE DE  
BRUXELLES

**RAPPORT D'ÉPREUVE INTÉGRÉE**

*CELESTIAL SPHERE*

Travail présenté par : *Weymeels Pierre*

en vue de l'obtention d'un bachelier en Informatique de gestion

Supervisé par : *Md. Herpigny Brigitte et Mr. Silovy Alain.*

*Année académique 2014 – 2015*



# Table des matières

<b>I</b>	<b>Préambule</b>	<b>6</b>
	Remerciements . . . . .	7
	Introduction . . . . .	8
	Résumé du sujet . . . . .	8
	Cadre de travail . . . . .	8
<b>II</b>	<b>Cycle de développement</b>	<b>9</b>
<b>1</b>	<b>Pré-étude</b>	<b>10</b>
1.1	Recueil des besoins . . . . .	10
1.1.1	Description du sujet . . . . .	10
1.1.2	Exigences fonctionnelles . . . . .	11
1.1.3	Exigences non fonctionnelles . . . . .	11
1.1.4	Cas d'utilisation . . . . .	12
1.1.4.1	Cas "Visionner la voûte céleste en mode équatorial" . . . . .	12
1.1.4.2	Cas "Visionner la voûte céleste en mode azimutal" . . . . .	13
1.1.4.3	Flots secondaires . . . . .	14
1.1.4.4	Cas "Initialisation de la position" . . . . .	15
1.1.4.5	Diagramme d'activités de la phase principale . . . . .	16
1.2	Analyse . . . . .	17
1.2.1	Diagramme de classes conceptuelles . . . . .	17
1.2.2	Dictionnaire de classes . . . . .	18
1.2.2.1	Description des dossiers . . . . .	18
1.2.2.2	Description du rôle des classes déduites du cas d'utilisation . . . . .	18
1.2.3	Base de données . . . . .	20
1.2.3.1	Table stars . . . . .	20
1.2.3.2	Table constellation . . . . .	21
1.2.3.3	Tables moonEphemeris et sunEphemeris . . . . .	22
<b>2</b>	<b>Elaboration</b>	<b>23</b>
2.1	Représentation de la voûte céleste . . . . .	23
2.2	Liaison avec OpenGL ES 2.0 . . . . .	26
2.2.1	Les classes de type contrôleur OpenGL . . . . .	26
2.2.2	Les classes de type rendu OpenGL . . . . .	27
2.3	Elaboration d'une classe de rendu pour OpenGL . . . . .	28
2.3.1	Le programme de liaison entre le CPU et le GPU . . . . .	28
2.4	Elaboration de la zone de vue sur la voûte . . . . .	28
2.5	Elaboration du repère équatorial . . . . .	29
2.6	Elaboration du repère azimutal . . . . .	29
2.7	Elaboration de l'affichage des noms . . . . .	31
2.7.1	La texture . . . . .	31
2.7.2	Le cadre d'affichage . . . . .	31
2.7.3	Le protocole de rendu . . . . .	31
2.8	Utilisation des éphémérides . . . . .	32
2.8.1	Interpolation de Bessel . . . . .	32
2.8.2	Précision de l'interpolation de Bessel . . . . .	33
2.9	Elaboration de la table "constellation" . . . . .	34
2.10	Les types de données à virgule flottante . . . . .	36
2.10.1	Conversion du type double en type float . . . . .	36
2.11	Liaison avec la base de données . . . . .	36

<b>3</b>	<b>Construction</b>	<b>37</b>
3.1	Phase d'initialisation	37
3.1.1	Diagramme d'interactions entre le modèle et l'activité de lancement	37
3.1.2	Construction de la classe StarsDome	40
3.1.3	Initialisation de la position d'un corps par interpolation des éphémérides	42
3.2	Phase principale	45
3.2.1	Diagramme d'initialisation de la phase principale	45
3.2.2	Diagramme d'initialisation de la GLSurface	47
3.2.3	Initialisation du repère azimutal	49
3.2.3.1	Diagramme d'initialisation	49
3.2.3.2	Implémentation de la matrice de transformation de repère	51
3.2.4	Construction de la fonctionnalité de rendu de noms	52
3.2.4.1	Activation de la classe de rendu des noms	52
3.2.4.2	Implémentation du VertexShader et du FragmentShader	54
3.2.4.3	Implémentation des méthodes utilisées pour le rendu des noms	55
3.3	Implémentation des shaders de la classe StarsRender	57
3.4	Implémentation de la méthode de changement de point de vue	58
3.5	Implémentation du contrôle de conversion double-float	60
<b>4</b>	<b>Transition</b>	<b>62</b>
4.1	Test utilisateur	62
4.1.1	Utilisation du GPS	62
4.1.2	Problème lié au rendu des noms	62
4.2	Tests unitaires	63
4.2.1	Tests unitaires dao	63
4.2.1.1	Constellation_daoTest	63
4.2.1.2	Stars_daoTest	63
4.2.2	Test unitaire de la classe Time_lib	65
4.2.3	Test unitaire de la classe Ephemeris	66
<b>III</b>	<b>Conclusion</b>	<b>68</b>
<b>IV</b>	<b>Annexe</b>	<b>70</b>
<b>A</b>	<b>Rendu final</b>	<b>71</b>
<b>B</b>	<b>Représentation des classes logicielles et des packages</b>	<b>75</b>
B.1	Remarques concernant le diagramme de classes logicielles :	75
B.2	Diagramme de classes logicielles	77
B.3	Dictionnaire de classes	78
B.3.1	Package activity	78
B.3.2	Package controller	79
B.3.3	Package model	80
B.3.4	Package dao	82
B.3.5	Package renderOpenGL	83
B.3.6	Package library	85
B.3.7	Package hac	86
<b>C</b>	<b>Améliorations futures</b>	<b>87</b>
C.1	Résolution des problèmes rencontrés	87
C.2	Amélioration du rendu visuel	87
C.3	Couplage avec les données d'orientations	87
C.4	Mode couplage orientation-position	88
<b>D</b>	<b>Android</b>	<b>89</b>
D.1	Description	89
D.2	Android et le langage de Java	89
D.3	La classe abstraite Context	89
D.4	La classe Activity	90
D.4.1	Description des méthodes	90
D.5	La classe "R"	90
D.6	Le dossier "res"	91

D.6.1	Description des sous-dossiers . . . . .	91
D.7	Le fichier "manifest" . . . . .	91
D.7.1	Description du manifest . . . . .	92
<b>E</b>	<b>OpenGL</b>	<b>93</b>
E.1	Description . . . . .	93
E.2	Les différentes versions . . . . .	93
E.3	Les primitives . . . . .	93
E.4	Rendering Pipeline . . . . .	94
E.4.1	Vertex Shader . . . . .	94
E.4.2	Tessellation . . . . .	95
E.4.3	Geometry Shader . . . . .	95
E.4.4	Transform Feedback . . . . .	95
E.4.5	Clipping (and "Face culling") . . . . .	95
E.4.6	Rasterization . . . . .	95
E.4.7	Fragment Shader . . . . .	95
E.4.8	Opération par échantillon . . . . .	96
E.5	Le pipeline de transformation . . . . .	96
E.5.1	Object coordinates . . . . .	96
E.5.2	Valeur d'homogénéité . . . . .	96
E.5.3	ModelView Matrix . . . . .	97
E.5.3.1	Matrice du modèle . . . . .	97
E.5.3.2	Matrice de la vue . . . . .	97
E.5.4	Projection Matrix . . . . .	98
E.5.4.1	Opération de projection . . . . .	98
E.5.5	Normalized Device Coordinates (ndc) . . . . .	98
E.5.6	Window Coordinates . . . . .	99
E.6	Le langage GLSL . . . . .	99
<b>F</b>	<b>Les repères de coordonnées</b>	<b>100</b>
F.1	Le repère géographique . . . . .	100
F.2	Le repère azimutal . . . . .	100
F.3	Le repère horaire . . . . .	101
F.4	Le repère équatorial . . . . .	102
<b>G</b>	<b>Définitions</b>	<b>103</b>
	<b>Bibliographie</b>	<b>107</b>

## Première partie

### Préambule

## Remerciements

Je tiens à adresser mes remerciements les plus sincères à l'ensemble de l'équipe pédagogique qui au cours de mes années d'études, m'a aidé à acquérir de nouvelles connaissances.

De plus, j'aimerais aussi particulièrement remercier mes superviseurs, Mr. Silovy Alain et Md. Herpigny Brigitte, pour leur disponibilité et leurs précieux conseils.

Mais surtout, je tiens à leur témoigner de ma profonde gratitude pour la grande qualité de leur enseignement et pour la compréhension qu'ils ont fait part, ainsi que leurs collègues, vis-à-vis des difficultés inhérentes qu'engendre l'association des études avec les obligations professionnelles.

Enfin, je tiens aussi à remercier tous mes proches pour leur soutien, et plus particulièrement ma compagne pour sa patience et sa gentillesse à mon égard.

# Introduction

” Il est grand temps de rallumer les étoiles ”<sup>1</sup>

De tous temps, l’homme a été fasciné par le ciel céleste.

Ainsi, il a cherché à le représenter, soit pour en améliorer sa propre compréhension, soit à des fins spirituelles ou artistiques.

La peinture de Vincent van Gogh intitulée ”La Nuit étoilée”, ou celle de William Herschel représentant la Voie Lactée<sup>2</sup>, en sont des exemples célèbres.

De nos jours, le ciel ne fascine plus autant qu’auparavant, et bien des raisons peuvent expliquer cette désaffection.

Une d’elles, qui à mon sens est la plus fondamentale, est liée à la pollution lumineuse du ciel.

En effet, cette pollution étant particulièrement élevée dans les zones urbanisées, fait disparaître la majeure partie des étoiles et, par conséquent, l’intérêt que l’on puisse apporter au ciel.

Outre mon intérêt personnel pour l’astronomie, il me semblait donc intéressant de fournir à un large public un outil convivial et didactique lui permettant de visualiser et de mieux comprendre le ciel céleste.

Ces 2 raisons ont guidé le choix de mon travail de fin d’études.

## Résumé du sujet

Le projet, consiste en la création d’un outil de représentation du ciel étoilé pouvant offrir aux utilisateurs la possibilité de se reconnecter avec leur environnement céleste.

Le ciel étoilé est modélisé par la voûte céleste qui est une sphère englobant la Terre et centrée sur elle. Et sur laquelle, les positions relatives des étoiles de notre galaxie appelée la ”Voie lactée”<sup>2</sup> sont projetées

Le but de ce projet consiste donc à représenter une telle voûte et cela, de la manière la plus fidèle possible.

De plus, afin que l’utilisateur puisse être aussi en contact direct avec son environnement céleste, cette application sera construite sur un support mobile.

Enfin, l’ensemble de ces propriétés, offrira donc la possibilité à un utilisateur de faire correspondre ce qu’il voit sur l’écran avec ce qu’il existe dans son environnement céleste.

## Cadre de travail

### Logique

La structure de conduite du projet est basée sur le modèle de ”Cycle de développement”<sup>3</sup>.

En ce qui concerne la structure d’analyse, elle est fondée sur la méthode appelée ”Processus unifié”<sup>4</sup>.

### Matériel

Un ordinateur personnel pour l’utilisation des logiciels et un smartphone Samsung Galaxy S3 GT-I9300 tournant sur la version 4.3 du système d’exploitation Android pour tester l’application.

### Logiciel

L’environnement de développement (IDE)<sup>5</sup> utilisé pour l’implémentation du code est la version 0.8.9 d’Android Studio qui inclut le kit de développement (SDK)<sup>6</sup> d’Android.

### Langage

Le langage ”Java” a été choisi comme langage de programmation et le ”Latex” pour la rédaction de ce rapport.

---

1. citation de Guillaume Apollinaire. 2. cf. définition en annexe G. 3. cf. bibliographie [17]. 4. cf. bibliographie [18] et [9]. 5. cf. définition en annexe G. 6. cf. définition en annexe G.



Deuxième partie

Cycle de développement

# Chapitre 1

## Pré-étude

### 1.1 Recueil des besoins

#### 1.1.1 Description du sujet

”Celestial Sphere” est une application Android<sup>1</sup> dont le but est de permettre à son utilisateur de découvrir la voûte céleste en manipulant l’interface graphique de son appareil mobile.

Cette application, est découpée en deux phases distinctes qui se succèdent :

**Phase d’initialisation :**

Dans cette phase, l’utilisateur est invité à faire un choix entre la possibilité d’initialiser sa position de manière manuelle ou automatique (via la puce GPS<sup>2</sup>), ou au contraire, de ne pas l’initialiser.

Après que le choix ait été effectué, une barre de progression renseigne l’état d’initialisation jusqu’au moment où ladite phase se termine.

**Phase principale :**

L’application propose initialement le repère de vue équatorial avec le pôle nord comme centre de rotation.

Cependant, dans le cas où la position de l’utilisateur serait initialisée, elle propose aussi le repère de vue azimutal ainsi que la possibilité de passer de l’un à l’autre à tout moment.

Néanmoins, et quel que soit le repère de vue, l’application permet de faire les actions suivantes :

- découvrir la voûte céleste en déplaçant son doigt sur l’écran.
- modifier le facteur d’agrandissement de la vue avec un mouvement à deux doigts.
- afficher ou masquer les noms des constellations et des étoiles.
- afficher ou masquer la grille équatoriale.
- se renseigner sur sa position actuelle.
- relancer la phase d’initialisation.

**Principes de fonctionnement :**

- les étoiles sont positionnées grâce aux données récoltées dans un catalogue de positions célestes.
  - la position de la Lune et du Soleil est obtenue par l’interpolation d’un fichier d’éphémérides.
  - la forme des constellations est déduite par le calcul des suites de positions célestes les caractérisant et contenues dans un fichier créé à cet effet.
  - les noms à afficher proviennent d’images qui sont positionnées sur un plan tangent à l’axe de vue de l’utilisateur.
  - pour le mode azimutal, l’application bascule le repère équatorial sur le repère azimutal, le centre de rotation devient par conséquent le zénith de l’utilisateur.
- En ce qui concerne la correspondance entre les deux repères, la mise à jour de l’environnement est assurée par le calcul du temps sidéral local .

---

1. cf. définition en annexe G.    2. cf. définition en annexe G.

### 1.1.2 Exigences fonctionnelles

#### Détails concernant l'interface graphique :

- affiche les étoiles visibles à l'oeil nu de magnitude apparente<sup>3</sup>  $\leq 5$
- affiche la position de la Lune et du Soleil
- en mode portrait, affiche un angle de vue initial de 90 degrés sur

$$\frac{l}{L} \times 90$$

degrés où "L" et "l" correspondent respectivement à la longueur et la largeur de l'écran.

En mode paysage, cet angle de vue initial sera exactement l'inverse.

- affiche la forme d'une partie des principales constellations.
- affiche la grille équatoriale pour favoriser l'effet voûte (sur demande).
- affiche le nom des constellations et de ses étoiles possédant un nom propre (sur demande).
- affiche l'horizon et une partie du méridien de l'utilisateur (en mode azimutal).
- renseigne sur la position de l'utilisateur (sur demande).

### 1.1.3 Exigences non fonctionnelles

**Utilisabilité :** intuitif, ne demande pas de formation spécifique.

**Performance :** Fonctionnel en moins de 15 secondes après le lancement de l'application.

**Fiabilité :** Précision de 0.001 degré d'angle sur la position de la Lune et du Soleil.

**Disponibilité :** 24h/24 7j/7.

**Sécurité :** Pas de données ou d'actions sensibles.

**Portabilité :** Système Android, version  $\geq 4.0$ .

**Maintenabilité :**

Utilisable jusqu'au 3 novembre 2021 (dernière date possible pour l'interpolation des éphémérides).

---

3. cf. définition en annexe G.

### 1.1.4 Cas d'utilisation

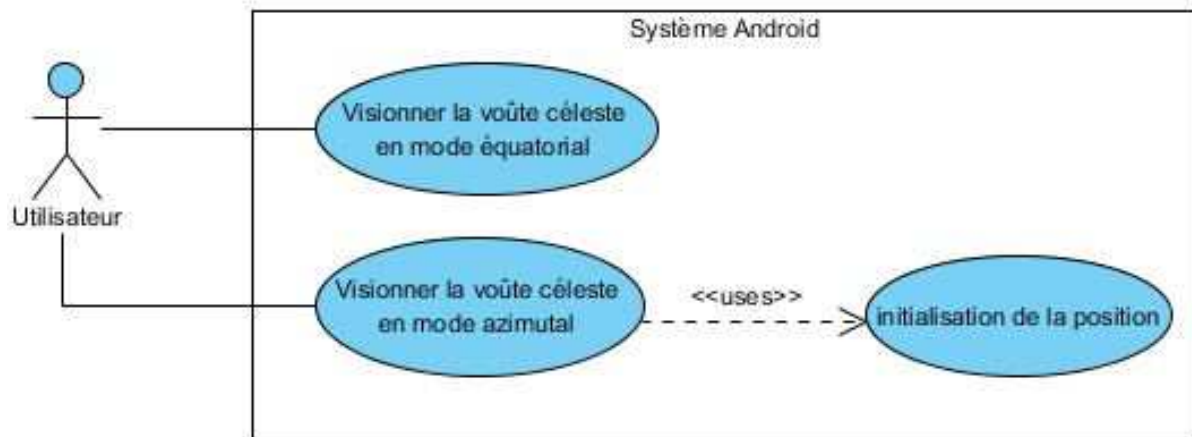


FIGURE 1.1 – Diagramme de cas d'utilisation.

#### 1.1.4.1 Cas "Visionner la voûte céleste en mode équatorial"

**Précondition :** La phase d'initialisation s'est effectuée sans erreur.

**Flot principal :**

TABLE 1.1 – Description du cas "Visionner la voûte céleste en mode équatorial".

Utilisateur	Système Android
<p><b>Événement secondaire :</b> L'utilisateur effectue une action secondaire.(cf. flots secondaires.)</p> <p><b>4 (événement principal) :</b> L'utilisateur glisse son (ou ses) doigt(s) sur l'écran.</p>	<p><b>1 :</b> Le système calcule la partie initiale du repère équatorial et démarre le processus de mise à jour des positions des corps célestes (TimerCss).</p> <p><b>2 :</b> Le système affiche la partie initiale de la voûte.</p> <p><b>3 :</b> Le système se place en attente d'un événement.</p> <p><b>TimerCss événement :</b> Mise à jour des positions. Le système rafraîchit son interface graphique.</p>
	<p><b>5 :</b> Le système calcule soit la partie à afficher en fonction du mouvement de translation sur l'écran, soit le nouveau facteur d'agrandissement de la vue.</p> <p><b>6 :</b> Le système rafraîchit son interface graphique en fonction du calcul précédent et se place à nouveau en attente d'un événement.</p>

#### 1.1.4.2 Cas "Visionner la voûte céleste en mode azimutal"

**Précondition :** La position à été initialisée et l'utilisateur a choisi d'activer le repère azimutal.

**Flot principal :**

TABLE 1.2 – Description du cas "Visionner la voûte céleste en mode azimutal".

Utilisateur	Système Android
<p><b>Événement secondaire :</b> L'utilisateur effectue une action secondaire.(cf. flots secondaires.)</p> <p><b>4 (événement principal) :</b> L'utilisateur glisse son (ou ses) doigt(s) sur l'écran.</p>	<p><b>1 :</b> Le système calcule et effectue la matrice de transformation du repère équatorial en un repère azimutal et démarre le processus de mise à jour de cette matrice de transformation (timerRaz).</p> <p><b>2 :</b> Le système démarre le processus de mise à jour de cette matrice de transformation (timerRaz).</p> <p><b>3 :</b> Le système affiche le repère azimutal.</p> <p><b>4 :</b> Le système se place en attente d'un événement.</p> <p><b>TimerCss événement :</b> Mise à jour des positions des corps du système solaire (cf. le cas d'utilisation du repère équatorial). Le système rafraîchit son interface graphique.</p> <p><b>TimerRaz événement :</b> Mise à jour de la position relative de la voûte vis-à-vis du repère azimutal. Le système rafraîchit son interface graphique.</p> <p><b>5 :</b> Le système calcule soit la partie à afficher en fonction du mouvement de translation sur l'écran, soit le nouveau facteur d'agrandissement de la vue.</p> <p><b>6 :</b> Le système rafraîchit son interface graphique en fonction du calcul précédent et se place à nouveau en attente d'un événement.</p>

### 1.1.4.3 Flots secondaires

**Précondition :** Le flot principal d'un des cas de visualisation est en cours.

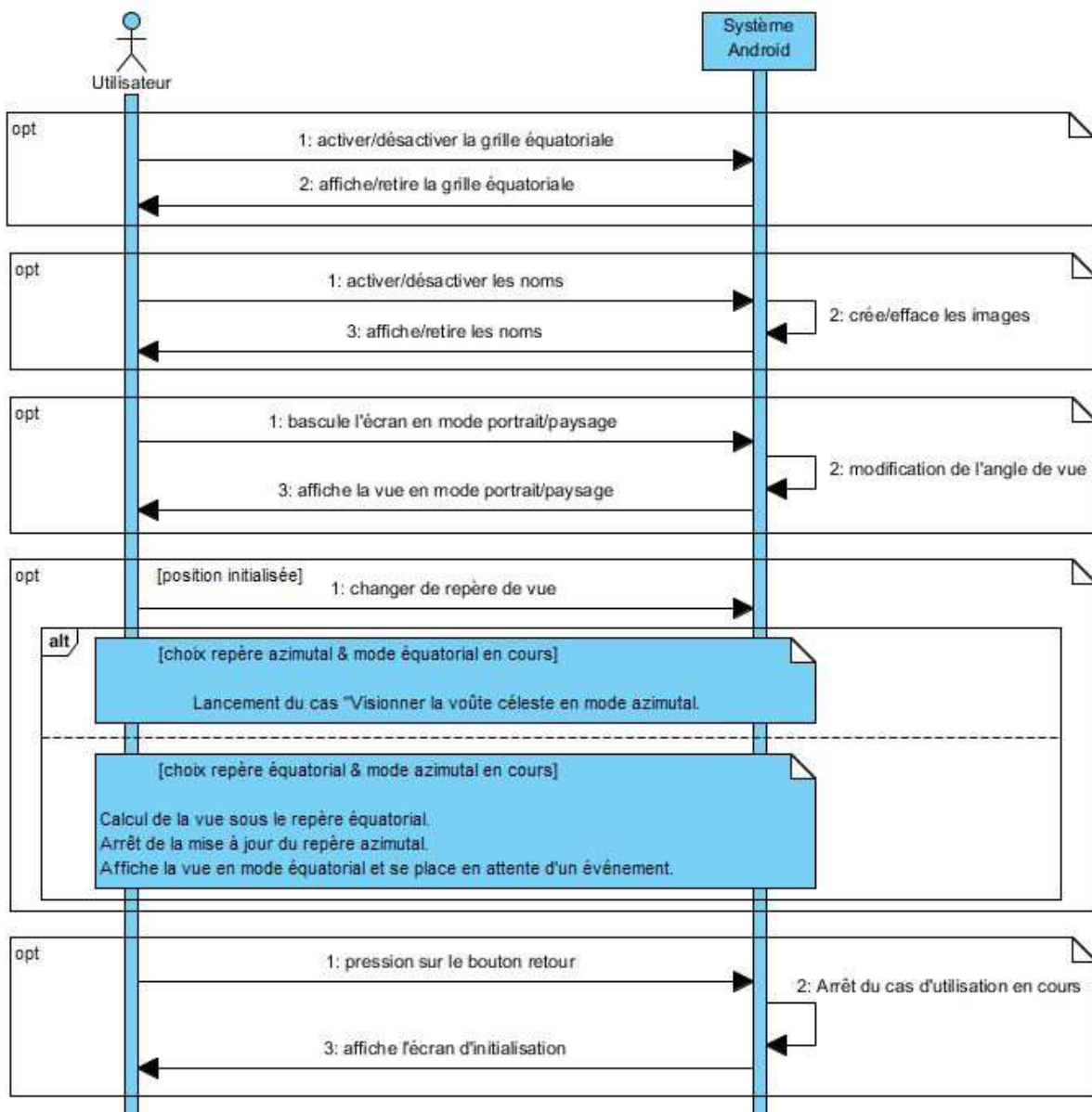


FIGURE 1.2 – Choix optionnels.

**Légende** => [ ] : représente une condition à remplir. | alt : représente une alternative. | opt : un déroulement optionnel.

#### 1.1.4.4 Cas "Initialisation de la position"

**Précondition :** En début de phase d'initialisation :

- soit par lancement de l'application ;
- soit après que l'utilisateur aie appuyé sur le bouton "retour".

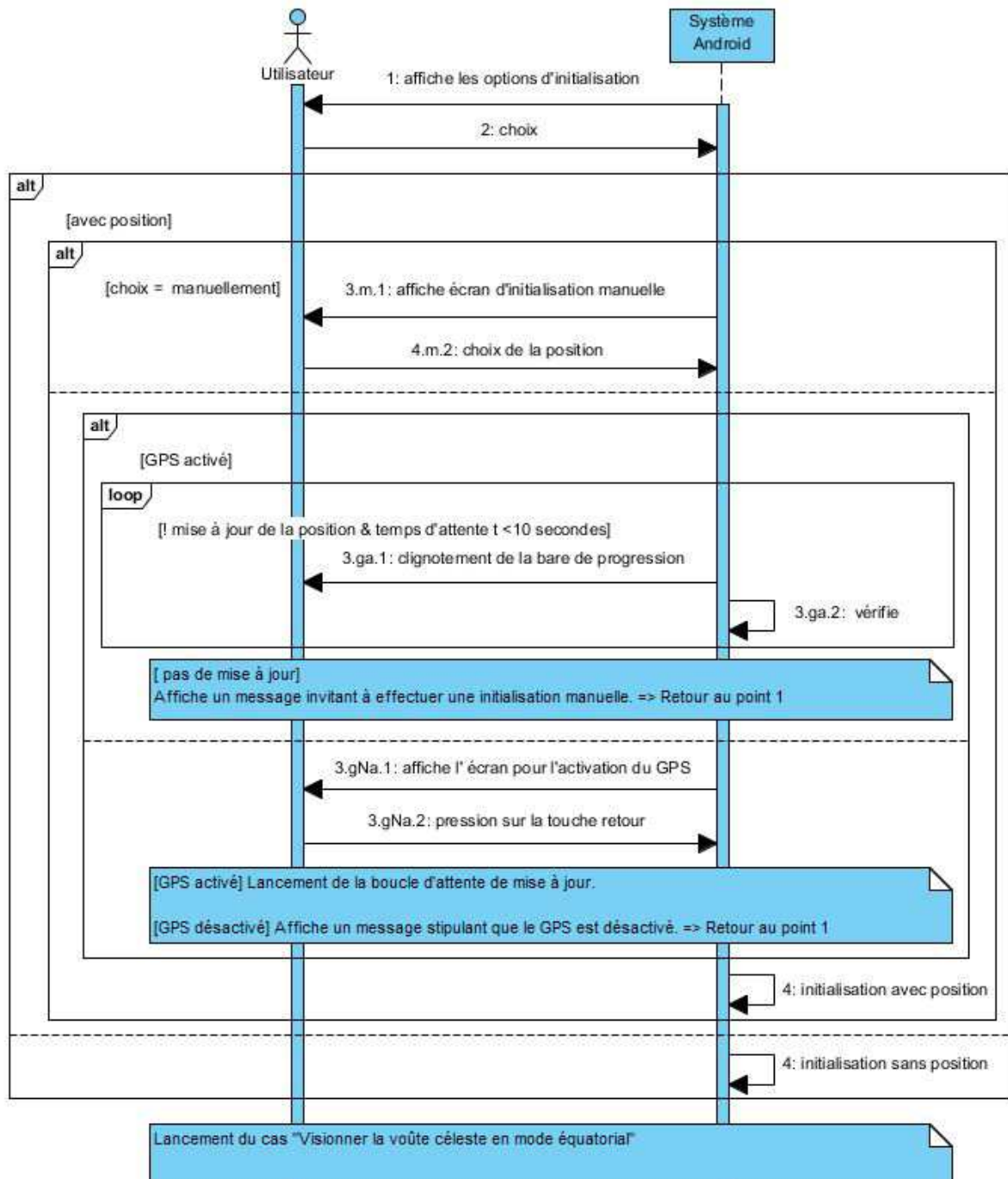


FIGURE 1.3 – Initialisation de la position.

**Légende** => m : signifie manuel. | ga : signifie GPS activé. | gNa : signifie GPS non activé.

#### 1.1.4.5 Diagramme d'activités de la phase principale

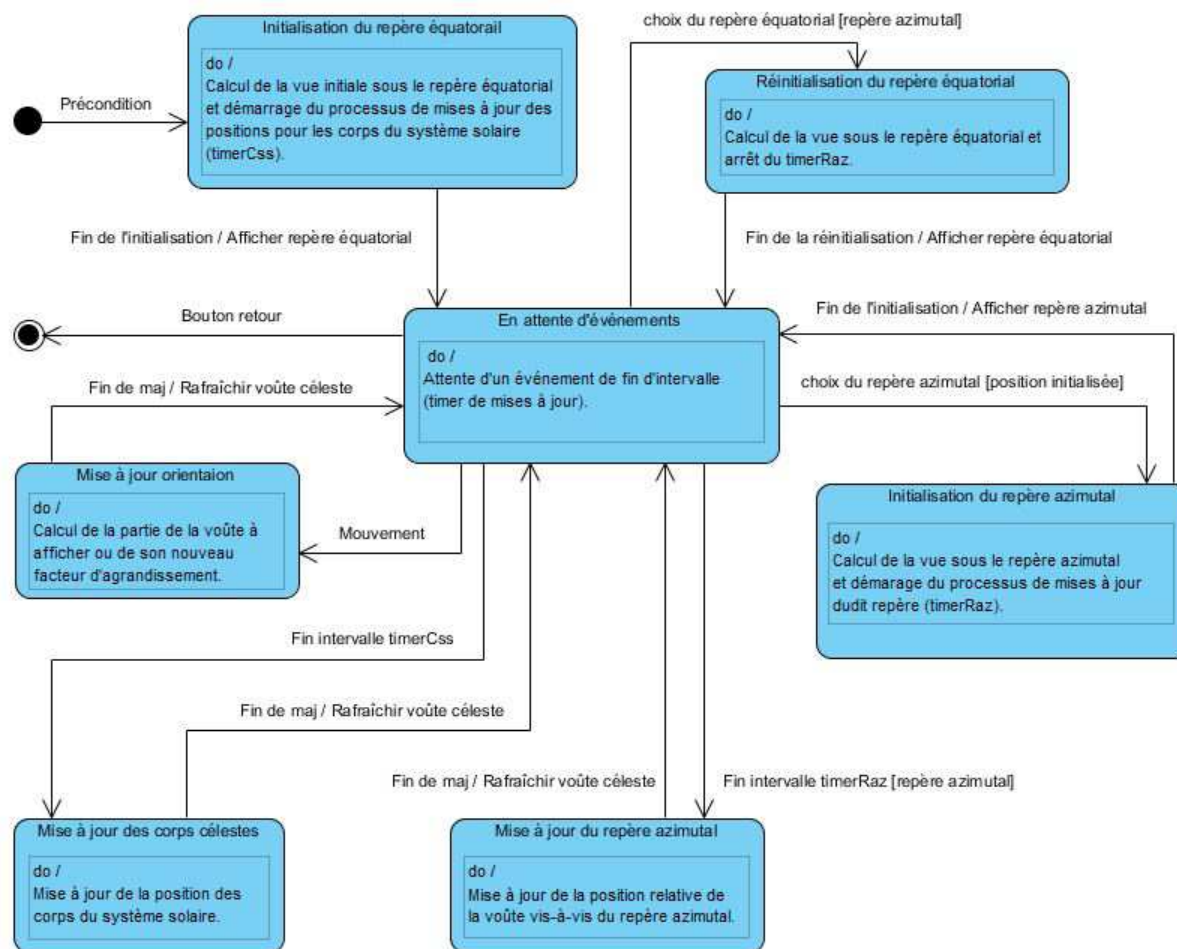


FIGURE 1.4 – Diagramme d'activités.

**Légende** => le point noir représente le début et celui entouré d'un cercle la fin de la phase principale.



## 1.2 Analyse

### 1.2.1 Diagramme de classes conceptuelles

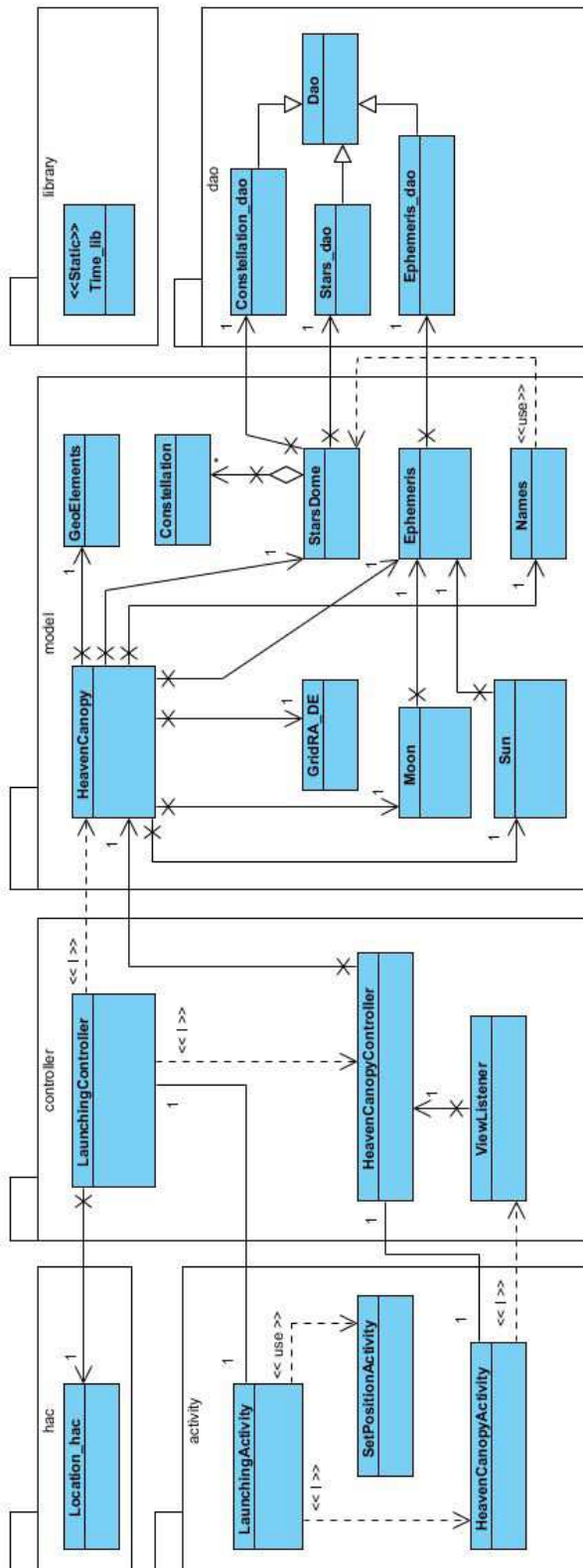


FIGURE 1.5 – Classes conceptuelles.

**Légende** ==> I signifie "instancie". | use : signifie utilise. | les chiffres : le nombre d'instance concernée par le lien. | une croix : sens interdit du côté où cette dernière est indiquée.

## 1.2.2 Dictionnaire de classes

### 1.2.2.1 Description des dossiers

**activity :**

Dossier regroupant les classes dédiées à la gestion de l’affichage de l’interface utilisateur.

**model :**

Dossier regroupant les classes dédiées à la gestion des données manipulées par l’application.

**controller :**

Dossier regroupant les classes dédiées au contrôle de l’initialisation du système et à la transmission des événements provenant de l’interface graphique.

**Data Access Object "dao" :**

Dossier regroupant les classes permettant l’accès à la base de données.

**Hardware Accès Controller "hac" :**

Dossier regroupant les classes permettant l’accès aux composants utilisés par l’application.

### 1.2.2.2 Description du rôle des classes déduites du cas d’utilisation

**Dossier "model" :**

**HeavenCanopy :**

Assure l’initialisation et la gestion de l’ensemble des éléments constituant la voûte céleste.

**StarsDome :**

Cette classe fournit la position des étoiles et initialise les différentes constellations.

**GridRA\_DE :**

Cette classe construit la grille des parallèles de déclinaisons et des méridiens d’ascensions droites.

**Constellation :**

Cette classe contient les paramètres propres à une constellation (nom, index de branche, séquence d’étoiles).

**Ephemeris :**

Contient les méthodes fournissant la position du corps céleste : interpolation de Bessel des éphémérides,...

**Moon :**

Initialise et contient les paramètres propres à ce corps céleste (position,couleur...).

**Sun :**

Initialise et contient les paramètres propres à ce corps céleste (position,couleur...).

**Names :**

Contient les liens vers les ressources nécessaires à l’affichage des noms et leur position.

**GeoElements :**

Contient les éléments géographiques (méridien,horizon,...) et la position de l’utilisateur (latitude,longitude).

**Dossier "activity" :**

**LaunchingActivity :**

Lance une interface graphique composée d’une barre permettant de suivre la progression de la phase d’initialisation du système.

**SetPositionActivity :**

Lance une interface graphique permettant à l’utilisateur d’initialiser sa position manuellement.

**HeavenCanopyActivity :**

Lance l’interface graphique représentant la voûte céleste.

**Dossier "controller" :**

**LaunchingController :**

Lance l’initialisation du modèle et renseigne LaunchingActivity de l’état d’avancement de cette initialisation.

**HeavenCanopyController :**

Initialise et contrôle le "timer" de mise à jour des positions de la Lune et du Soleil.

**GLSurfaceViewListener :**

Interprète et transmet les événements provenant de la vue représentant la voûte céleste.

**Dossier "dao" :**

**Stars\_dao :**

Classe permettant l'accès à la base de données SQLite.

**Constellation\_dao :**

Classe permettant l'accès à la table "constellation" de la base de données SQLite.

**Stars\_dao :**

Classe permettant l'accès à la table "stars" de la base de données SQLite.

**Ephemeris\_dao :**

Classe permettant l'accès aux tables "moonEphemeris" et "sunEphemeris" de la base de données SQLite.

**Dossier "library" :**

**Time :**

Contient les méthodes fournissant les différents temps utiles à l'application.

**Dossier "hac" :**

**Location\_hac :**

Contient les méthodes permettant l'accès aux valeurs de position de la puce GPS.

### 1.2.3 Base de données

La base de données de type SQLite<sup>4</sup> de l'application portera le nom de "dbStarsTracker" et elle sera située dans un de ses dossiers.

#### 1.2.3.1 Table stars

Cette table fournit la position des étoiles selon l'origine des temps J2000<sup>5</sup>.

**Description :** La table dont la structure physique est située ci-dessous contient 1628 entrées et provient de la fusion entre trois tables complémentaires fournies par le centre de données astronomiques de Strasbourg (CDS)<sup>6</sup> via la liaison sur leur identifiant (HR)(commun pour les trois fichiers).

**"Bright Star Catalogue (BSC)" :**

ce fichier<sup>7</sup> a été restreint aux valeurs de magnitude plus petite ou égale à 5, et aux colonnes HR, RAJ2000<sup>8</sup>, DEJ2000<sup>9</sup>, VMAG<sup>10</sup>.

Ce qui nous donne 1628 entrées.

**"Catalogue of the Brightest Stars" (The catalogue) :**

ce fichier<sup>11</sup> a été restreint aux colonnes HR, Bayer<sup>12</sup>, Const<sup>13</sup>, Dist<sup>14</sup> et possède 1628 entrées.

**"Catalogue of the Brightest Stars" (Star names) :**

ce fichier<sup>11</sup> a été restreint aux colonnes "HR" et "Name"<sup>15</sup> et possède 80 entrées.

**Champs :**

**HR :** numéro d'identification de l'étoile .

**name :** nom propre de l'étoile .

**grekletter :** lettre grecque associée à l'étoile .

**constellation :** nom de la constellation associée à l'étoile .

**RAJ2000 :**

ascension droite en heure-minute-seconde au format texte<sup>16</sup> de l'étoile à la date J2000.

**DEJ2000 :**

déclinaison en degré-minute-seconde au format texte<sup>16</sup> de l'étoile à la date J2000.

**magnitude :** luminosité de l'étoile.

**distance :** distance à laquelle est située l'étoile (en année-lumière).









Stars		
 <b>HR</b>	<b>integer(4)</b>	
 <b>name</b>	<b>varchar(20)</b>	<b>N U</b>
 <b>grekletter</b>	<b>varchar(5)</b>	<b>N</b>
 <b>constellation</b>	<b>varchar(3)</b>	<b>N</b>
 <b>RAJ2000</b>	<b>varchar(10)</b>	
 <b>DEJ2000</b>	<b>varchar(9)</b>	
 <b>magnitude</b>	<b>real(3)</b>	
 <b>distance</b>	<b>integer(5)</b>	

FIGURE 1.6 – Table contenant les données relatives aux étoiles.

**Légende =>** La clef signifie que le champ est un identifiant unique qui joue le rôle de clef primaire pour la table . | N : signifie que la valeur peut être nulle. | U signifie que la valeur doit être unique si elle existe.

4. moteur de base de données relationnelle (SGBDR) accessible par le langage SQL, définitions en annexe G. 5. cf. définition en annexe G. 6. cf. bibliographie [5]. 7. cf. bibliographie [12]. 8. ascension droite en J2000, voir annexe F.4. 9. déclinaison en J2000, voir annexe F.4. 10. luminosité de l'étoile (magnitude visuelle), définition en annexe G. 11. cf. bibliographie [11]. 12. lettre grecque attribuée à l'étoile. 13. abréviation de la constellation à laquelle appartient l'étoile. 14. distance à laquelle se situe l'étoile (année-lumière), définition en annexe G. 15. nom propre de l'étoile. 16. description en annexe F.4.

### 1.2.3.2 Table constellation

**Description :** La table dont la structure physique est située ci-dessous, a été élaborée à partir des données fournies par la table stars et des descriptions fournies par le site de l'union astronomique internationale (IAU) <sup>17</sup>.

**Champs :**

**name :** le nom de la constellation (en trois lettres).

**stars\_sequence :** la suite d'étoiles permettant le dessin de la constellation.

**branch\_index :**

la suite d'index de stars\_sequence. Chaque index correspond au départ d'une nouvelle branche de la constellation.

**RA\_center :** ascension droite en heures et décimales du centre de la constellation.

**DE\_center :** déclinaison en degrés et décimales du centre de la constellation.






constellation	
 <b>name</b>	<b>varchar(3)</b>
 stars_sequence	varchar(255)
 branch_index	varchar(255)
 RA_center	real(7)
 DE_center	real(7)

FIGURE 1.7 – Table contenant les données relatives aux constellations.

**Légende** => La clef signifie que le champ est un identifiant unique qui joue le rôle de clef primaire pour la table .

---

17. cf. bibliographie [3].

### 1.2.3.3 Tables moonEphemeris et sunEphemeris

**Description :** Les tables dont la structure physique est située ci-dessous, sont fournies par le site de "L'Institut de mécanique céleste et de calcul des éphémérides"<sup>18</sup>.

Elles fournissent la position en coordonnées équatoriales<sup>19</sup> pour le Soleil et la Lune (avec la phase de ce dernier), par pas de 12h, du 1 janvier 2015 à 0h au 4 novembre 2021 à 12h (5000 entrées).

**Champs :**

**DateUTC :** la date et l'heure en temps universel coordonné (UTC)<sup>20</sup>.

**RA :**

ascension droite en heure-minute-seconde au format texte du corps.

**DE :**

déclinaison en degré-minute-seconde au format texte du corps.

**phase :**

indique en degré la partie visiblement éclairée de la Lune pour un observateur sur Terre (0° correspond à la nouvelle Lune, 180° à la pleine lune).





moonEphemeris	
 <b>DateUTC</b>	<b>varchar(22)</b>
 RA	varchar(14)
 DE	varchar(14)
 phase	real(6)

FIGURE 1.8 – Table contenant les données relatives aux constellations.

**Légende** => La clef signifie que le champ est un identifiant unique qui joue le rôle de clef primaire pour la table .




sunEphemeris	
 <b>DateUTC</b>	<b>varchar(22)</b>
 RA	varchar(14)
 DE	varchar(14)

FIGURE 1.9 – Table contenant les données relatives aux constellations.

**Légende** => La clef signifie que le champ est un identifiant unique qui joue le rôle de clef primaire pour la table .

18. cf. bibliographie [6]. 19. description en annexe F.4. 20. cf. définition en annexe G.

# Chapitre 2

## Elaboration

### 2.1 Représentation de la voûte céleste

L'objectif de ce projet étant de présenter la voûte étoilée sur un écran, le premier réflexe a été de créer une image englobant l'ensemble de la voûte.

Hors, il est mathématiquement impossible de représenter une surface sphérique sur une surface plane de manière conforme<sup>1</sup> et équivalente<sup>2</sup>.

De plus, la distorsion provoquée augmente à mesure que la partie à modéliser s'étend.

Par conséquent, et après avoir étudié les différents types de projection, la projection stéréographique a été retenue car elle possède les propriétés suivantes :

**Angle :** les angles sont conservés, donc les formes et les directions aussi (projection conforme).

**Distance :** le facteur d'échelle s'accroît en s'écartant du centre.

**Aire :** équivalence au centre, mais une distorsion apparaît et s'accroît en s'écartant du centre.

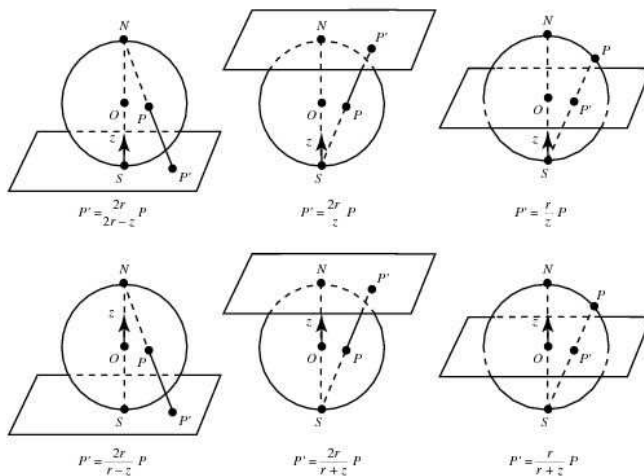


FIGURE 2.1 – Les différents types de projections stéréographiques.

**Légende** => Les formules mathématiques situées sous les images, correspondent au rapport de proportion entre la position du point et sa position projetée sur le plan. | "r" équivaut au rayon de la sphère, et z la coordonnée du point P selon l'axe qui passe par le point N et S.

**Image :** <http://mathworld.wolfram.com/StereographicProjection.html>

1. cf. définition en annexe G. 2. cf. définition en annexe G.

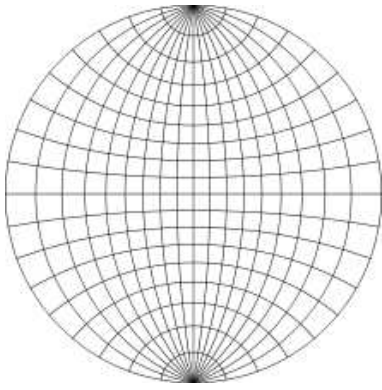


FIGURE 2.2 – Rendu des parallèles et des méridiens par projection stéréographique.

Image : <http://fr.wikipedia.org>

De ces propriétés, il apparaît nécessaire de créer en temps réel une projection stéréographique centrée sur l'axe de vue de l'utilisateur.

Une partie de la solution, serait apportée par l'utilisation de la bibliothèque de programmation graphique OpenGL ES disponible sur Android.

En effet, cette dernière apporte la possibilité de faire des transformations en temps réel.

Néanmoins, OpenGL ES ne propose pas le type de projection cité plus haut, mais propose le type perspective.

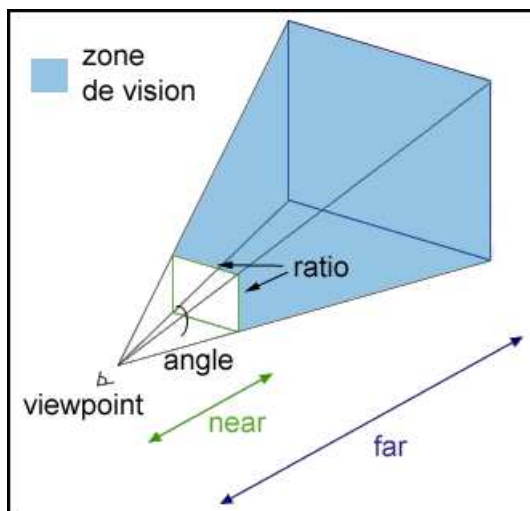


FIGURE 2.3 – Zone de vision d'une projection perspective.

Image : <http://openclassrooms.com/courses/creez-des-programmes-en-3d-avec-opengl>

Heureusement, une projection stéréographique équatoriale est équivalente à une du type perspective si celle-ci respecte les conditions suivantes (en assimilant la voûte à une sphère unité) :

1. L'axe de vue doit toujours passer par le centre de la sphère.
2. Le centre de projection perspective (position de l'œil) se situe toujours sur la surface.
3. L'axe de vue pointe en direction du point correspondant au centre de la zone visée sur la surface .
4. Le centre de visé se situe diamétralement opposé à l'œil.
5. Le plan proche de la perspective sera toujours celui de l'équateur de la sphère (c'est-à-dire, à une unité de l'œil).
6. Le plan éloigné de la perspective sera toujours celui du plan tangent à la sphère en son centre de visée (c'est-à-dire, à deux unités de l'œil).

Par le point 5 et 6, nous sommes assurés de pouvoir englober l'hémisphère opposé à l'œil, si on choisit un angle de vue de 90 degrés (la variation de cet angle permettant d'effectuer un zoom sur le centre de visée).



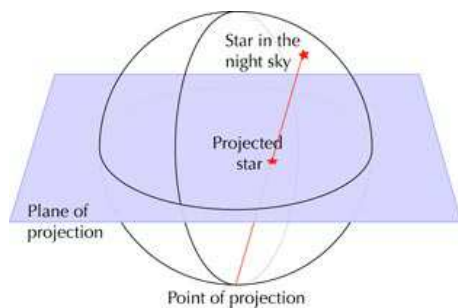


FIGURE 2.4 – Projection perspective résultant des conditions.

**Image :** <http://www.hps.cam.ac.uk/whipple/explore/astronomy/mapsoftheheavens/>

Pour conclure, ayant la volonté de ne pas trop limiter la gamme de version Android pouvant utiliser le programme et dans le souci d'utiliser une version moderne d'OpenGL ES, la version 2.0 à été choisie pour l'implémentation<sup>3</sup>.

---

3. voir en annexe E.2.

## 2.2 Liaison avec OpenGL ES 2.0

### 2.2.1 Les classes de type contrôleur OpenGL

OpenGL utilise un objet appelé "GLSurfaceView" comme surface de rendu OpenGL.

Ce dernier possède son propre fil d'exécution et utilise l'interface EGL <sup>4</sup> pour la liaison avec le système Android.

Cette surface, sera intégrée au layout xml "heavencanopy\_layout" déclarant l'interface graphique de HeavenCanopyActivity et déclarée comme variable d'instance de cette activité.

Cette vue OpenGL aura besoin d'une part, d'une classe implémentant l'interface "GLSurfaceView.Renderer" pour assurer le suivi des événements suivants :

**onSurfaceCreated(GL10 glUnused, EGLConfig config) :**

appelée à la création de la vue, elle permet l'initialisation des programmes de rendu graphique liés aux données fournies par le modèle.

**onSurfaceChanged(GL10 glUnused, int width, int height) :**

appelée lors de la création ou lors d'un basculement de l'écran.

**onDrawFrame(GL10 glUnused) :**

appelée à la création de la vue ou manuellement lors d'un changement du point de vue ou de la variation du zoom.

Ces méthodes, seront lancées sur un fil d'exécution de cette surface de rendu.

D'autre part, une classe appelée "GLSurfaceViewListener" et implémentant l'interface "GLSurfaceView.OnTouchListener" et "ScaleGestureDetector.OnScaleGestureListener" sera à l'écoute des événements suivants :

**onTouch(View view, MotionEvent motionEvent) :**

touché ou mouvement sur la vue.

**onDoubleTap(MotionEvent motionEvent) :**

double touché rapide sur la même zone de l'écran.

**onScaleBegin(ScaleGestureDetector scaleGestureDetector) :**

début d'un mouvement à deux doigts de type paire de ciseaux.

**onScale(ScaleGestureDetector scaleGestureDetector) :**

mouvement à deux doigts de type paire de ciseaux en cours.

Ce qui nous donne finalement l'ensemble des contrôleurs suivant :

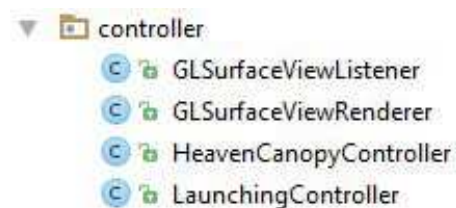


FIGURE 2.5 – Le package contrôleur.

---

4. cf. définition en annexe G.

Ainsi que les interfaces suivantes :

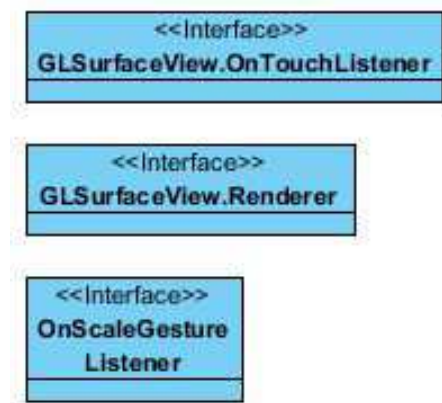


FIGURE 2.6 – Les interfaces liées à la surface de rendu openGl.

### 2.2.2 Les classes de type rendu OpenGL

Afin de séparer du modèle les méthodes liées à l'utilisation d'OpenGL, un package appelé "renderOpenGL" a été créé.

Dans ce package, on trouvera les classes responsables du rendu des différents objets du modèle :

- StarsRender ;
- ConstellationRender ;
- GridRA\_DERender ;
- SolarSystemBodyRender.
- NamesRender.
- GeoElementsRender.

Ces classes, hériteront de la classe abstraite "AbstractRender" pour l'implémentation de leurs méthodes communes.

Pour conclure, une classe responsable des opérations de rendu sera créé sous le nom de "RenderManager" et implémentera l'interface "RenderCommand" pour la déclaration de ses méthodes publiques.

Afin de permettre aux différents contrôleurs d'effectuer une action sur une classe de rendu en particulier, certaines méthodes posséderont un paramètre dont la valeur sera accessible via une énumération (appelée Render) des classes de type rendu.

Ce qui nous donne finalement le package suivant :

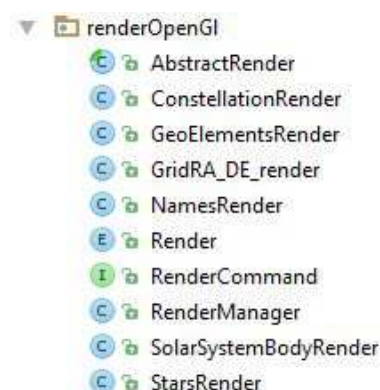


FIGURE 2.7 – Le package contrôleur.

## 2.3 Elaboration d'une classe de rendu pour OpenGL

Ce type de classe assure les responsabilités suivantes :

- la possession des données à transmettre au pipeline graphique sous la forme de buffer ou de tableau dont le type est de simple précision (float) car OpenGL ES 2.0 ne supporte pas la double précision<sup>5</sup> ;
- la possession des matrices de transformation à appliquer aux données ;
- l'implémentation en langage GLSL des programmes "Vertex Shader" et du "Fragment Shader"<sup>6</sup> (étapes du pipeline graphique) contenue sous la forme de type String ;
- la création, l'initialisation, l'activation et la désactivation d'un programme pour chaque shader ;
- la création, l'initialisation, l'activation et la désactivation d'un programme responsable de la liaison entre le CPU et le GPU.
- la transmission au pipeline graphique et aux shaders des données (positions, couleur(s), textures,...) à l'intérieur dudit programme.

### 2.3.1 Le programme de liaison entre le CPU et le GPU

**Initialisation** Lors de l'initialisation du programme de liaison, on lui attribuera le Vertex Shader et "Fragment Shader", tous les deux ayant été préalablement initialisés avec leur code d'implémentation respectif et compilé.

**Exécution** Lorsque le fil d'exécution OpenGL l'exécute (CPU), il initialise les variables et les constantes des shaders et assure la transmission des mémoires tampons (buffers) au pipeline graphique du GPU où ces derniers seront traitées (à un certain niveau du pipeline) par l'exécution sous le GPU du Vertex Shader et Fragment Shader.

## 2.4 Elaboration de la zone de vue sur la voûte

Pour simuler un point de vue, OpenGL applique une matrice de transformation appelée la matrice de vue<sup>7</sup>.

Pour élaborer cette matrice, nous avons besoin de définir les vecteurs suivants :

1. Le vecteur représentant la position de l'oeil (eyeXyzPosition) ;
2. Le vecteur représentant la direction de l'axe de vue (viewXyzDirection) ;
3. Le vecteur représentant la direction de l'axe perpendiculaire à l'axe de vue (upXyzDirection) ;

Ces vecteurs, seront de dimension trois et seront définis selon un repère de coordonnées cartésiennes x, y et z, dont l'origine, sera le centre de la sphère de rayon unité qui représente la voûte céleste.

De plus, nos vecteurs posséderont les propriétés suivantes :

- "eyeXyzPosition" sera toujours situé sur la surface de la sphère unité à l'exacte opposé du centre de visé.
- "viewXyzDirection" représentera le vecteur de position du centre de visé qui sera toujours situé sur la surface de la sphère.
- "upXyzDirection" sera calculé de sorte à toujours pointer dans la direction et le sens du pôle nord pour le repère équatorial, et dans la direction et le sens du zénith pour le repère azimutal.

Pour rappel, les propriétés des deux premiers vecteurs ci-dessus sont imposées par la volonté de créer une projection stéréographique à l'aide d'une projection de type perspective fournie par OpenGL.

Enfin, la matrice de vue<sup>7</sup> sera construite avec une méthode fournie par OpenGL qui prend en paramètre les vecteurs définis ci-dessus.

Pour conclure, nous attribuerons à la classe "RenderManager" la responsabilité de l'initialisation et de la mise à jour, et à la classe "AbstractRender" celle de la détention et du partage de la matrice de vue.

### Calcul de la nouvelle zone de vue suite à un mouvement

Lorsqu'un mouvement de translation sera détecté sur l'écran, une méthode sera chargée de faire correspondre à ce nouveau centre d'écran, le centre de visé adéquat de la sphère céleste<sup>8</sup>.

5. <https://software.intel.com/en-us/android/articles/porting-opengl-games-to-android-on-intel-atom-processors-part-2>

6. voir en annexe E.4.1 et E.4.7. 7. voir en annexe E.5.3.2. 8. voir le chapitre 3.4.

## 2.5 Elaboration du repère équatorial

Les positions des différents objets pouvant être représentés dans ce repère (la grille RA-DE, les étoiles, la Lune et le Soleil) seront initialisées dès leur création vis-vis du repère équatorial et selon les coordonnées cartésiennes géocentriques (X,Y,Z) situées sur une sphère de rayon unité représentant la voûte céleste (cf. l'image ci-dessous).

En ce qui concerne les constellations, et afin de faciliter le calcul, les liens entre les étoiles seront des cordes sur cette sphère.

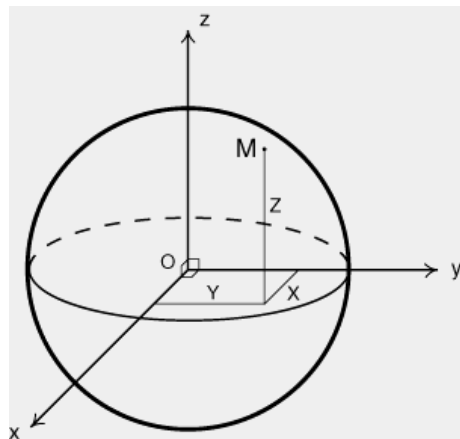


FIGURE 2.8 – Coordonnées cartésiennes d'un point sur une sphère.

Image : <http://seig.ensg.ign.fr/>

Pour conclure, les noms seront représentés sur des plans tangents à la sphère et le haut de la vue sera toujours dirigé en direction du pôle nord, ce qui offrira la possibilité de glisser le long des différentes latitudes.

## 2.6 Elaboration du repère azimutal

Le repère azimutal<sup>9</sup> décrit l'environnement de l'utilisateur.

Les éléments le caractérisant sont l'horizon, le méridien<sup>10</sup>, le zénith et les points cardinaux.

### Valeurs nécessaires à l'élaboration

Afin de transformer le repère équatorial en un repère azimutal, il est important d'obtenir les valeurs suivantes :

#### Le temps sidéral local :

correspond à l'angle qui sépare le point vernal<sup>11</sup> et le méridien du lieu géographique d'observation.

Cet angle correspond à l'addition de l'ascension-droite  $\alpha^{11}$  avec l'angle horaire  $H^{12}$ .

Mais aussi, en additionnant la longitude du lieu d'observation  $\lambda$  avec le temps sidéral à Greenwich  $T_G^{13}$ .

#### La longitude et la latitude :

au sein de cette application, on utilisera la longitude pour le calcul du temps sidéral local, et la latitude  $\phi$  pour obtenir l'angle de colatitude  $\Phi^{12}$ .

### Elaboration du temps sidéral local

Les coordonnées géographiques étant obtenues soit par le GPS, soit par une introduction manuelle de l'utilisateur, il reste donc, à calculer la valeur du temps sidéral local.

Ce dernier s'obtient par l'algorithme suivant :

1. on récupère le temps universel coordonné (UTC)<sup>14</sup> courant ;
2. on calcule la date julienne<sup>15</sup> courante ;

9. voir annexe F.2 10. cf. définition en annexe G. 11. voir annexe F.4 12. cf. figure 2.9. 13. cf. figure 2.10. 14. cf. définition en annexe G. 15. cf. définition en annexe G.

3. on calcule le temps sidéral à Greenwich  $T_G$  avec la date julienne comme paramètre ;
4. on additionne à  $T_G$  la valeur de la longitude du lieu dans le cas d'une longitude située à l'est du méridien de Greenwich et on effectue une soustraction dans le cas contraire.

Cette dernière opération nous donne la valeur du temps sidéral local.

## La transformation du repère

L'algorithme de transformation du repère équatorial en un repère azimutal est expliqué dans le chapitre consacré à la construction de l'application <sup>16</sup>.

Pour conclure, la transformation peut être facilement inversée, mais il est moins coûteux algorithmiquement de revenir à la matrice du modèle <sup>17</sup> initial. C'est-à-dire la matrice qui n'applique aucune transformation (matrice unité).

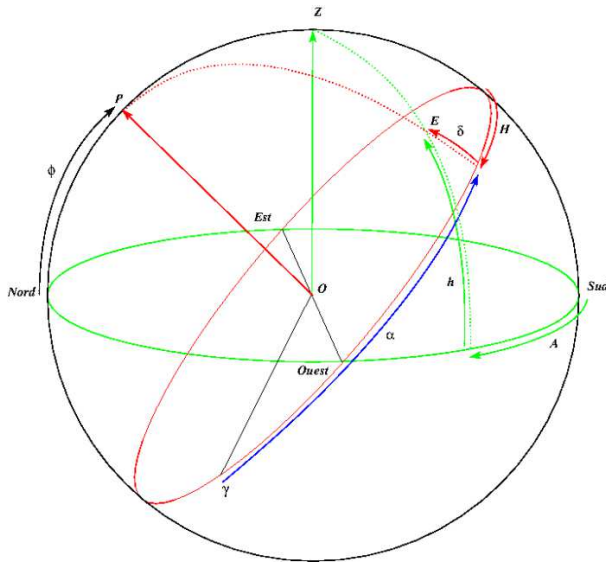


FIGURE 2.9 – Relation entre le repère azimutal et le repère équatorial.

**Légende** =>  $\gamma$  : le point vernal. |  $\alpha$  : ascension-droite. |  $\delta$  : déclinaison. |  $H$  : angle horaire. |  $E$  : une étoile. |  $A$  : azimut. |  $h$  : hauteur. |  $Z$  : zénith. |  $P$  : pôle nord. |  $\Phi$  : colatitude ( $90^\circ$  - latitude).

**Image** : [http://media4.obspm.fr/public/AAM/pages\\_trigo/impression.html](http://media4.obspm.fr/public/AAM/pages_trigo/impression.html)

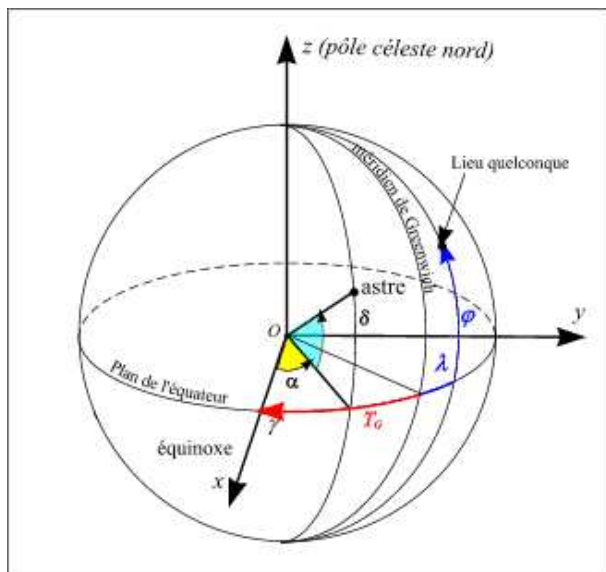


FIGURE 2.10 – Relation entre le repère équatorial et la position géographique d'un observateur.

**Légende** =>  $\gamma$  : le point vernal. |  $\alpha$  : ascension-droite. |  $\delta$  : déclinaison. |  $T_G$  : temps sidéral à Greenwich. |  $\lambda$  : longitude. |  $\phi$  : latitude.

**Image** : [http://www.imcce.fr/vt2004/en/fiches/fiche\\_n05\\_08\\_eng.html](http://www.imcce.fr/vt2004/en/fiches/fiche_n05_08_eng.html)

16. cf. chapitre de construction 3.2.3.2. 17. voir annexe E.5.3.1.

## 2.7 Elaboration de l’affichage des noms

Pour afficher du texte à l’intérieur de l’environnement graphique d’OpenGL, nous devons pour chaque ensemble de lettres à d’afficher obtenir les éléments suivants :

### 2.7.1 La texture

L’image possédera un fond transparent et sa taille sera adaptée au texte.

De plus, le type choisi doit assurer une certaine qualité sans être trop encombrante d’un point de vue de la mémoire.

Avec l’aide d’un programme créé à cet effet, et en respectant les propriétés ci-dessus, on obtient 48 images pour les étoiles et 32 images pour les constellations.

Le type choisi est le ”png” et leur poids vaut 260 octets en moyenne.

Elles ont été placées dans le dossier contenant les ressources (drawable-nodpi) de l’application.

### 2.7.2 Le cadre d’affichage

L’ensemble des positions, utilisé pour placer l’image dans l’environnement en trois dimensions doit être choisi avec précaution.

En effet, pour OpenGL l’image possède une face avant et une face arrière.

Si aucune des deux faces n’est présentées perpendiculairement à l’axe de vue, la représentation de l’image située sur l’axe subirait au minimum une altération et au pire un effacement.

Par conséquent, il est important de choisir le plan adéquat pour le calcul du cadre de positions.

Néanmoins, afin de permettre que le texte soit affiché en bas à droite de l’élément à nommer, et cela sans trop augmenter la complexité de calcul, le plan perpendiculaire à cet élément a été choisi au lieu de celui perpendiculaire au point représentant le centre du texte.

### 2.7.3 Le protocole de rendu

Pour permettre le rendu du cadre de positions, il faut que le cadre soit défini par deux primitives de type triangle dont les sommets seront gérés par le pipeline graphique selon un ordre précis.

Par défaut, la face avant d’un triangle est déterminée par le sens contraire des aiguilles d’une montre.

De plus, pour que l’image soit correctement placée sur la zone de mappage (le cadre), il faut que ses sommets soient gérés en même temps par les shaders et dans le même ordre.

L’ordre des sommets choisis (pour le cadre et l’image) est le suivant :

#### **Pour le premier triangle :**

1. le coin en bas à gauche ;
2. le coin en bas à droite ;
3. le coin en haut à droite.

#### **Pour le deuxième triangle :**

1. le coin en haut à droite ;
2. le coin en haut à gauche ;
3. le coin en bas à gauche.

Pour conclure, étant donné que le cadre sera choisi à chaque fois sur un plan tangent à la sphère, et que les positions des sommets seront définis en fonction d’un point de vue situé au centre de la sphère, la face arrière des triangles ne sera donc jamais visible.

## 2.8 Utilisation des éphémérides

Pour rappel, les tables d'éphémérides utilisées au sein de cette application, fournissent la position du corps concerné à 0h et 12h pour chaque jour.

Par conséquent, par le terme "date" il sera fait ici allusion à un moment précis d'un jour. Deux dates différentes de la table pouvant correspondre au même jour !

DateUTC	RA	DE	phase
2015-01-01T00:00:00.00	3 15 2.89467	+15 3 2.1255	49.57
2015-01-01T12:00:00.00	3 41 57.29664	+16 13 7.3856	43.56
2015-01-02T00:00:00.00	4 8 56.42348	+17 10 1.9054	37.6
2015-01-02T12:00:00.00	4 35 57.57323	+17 53 9.6704	31.72
2015-01-03T00:00:00.00	5 2 57.23332	+18 22 7.7496	25.91
2015-01-03T12:00:00.00	5 29 51.32855	+18 36 47.1342	20.2
2015-01-04T00:00:00.00	5 56 35.52012	+18 37 13.0106	14.65
2015-01-04T12:00:00.00	6 23 5.52534	+18 23 44.3614	9.44
2015-01-05T00:00:00.00	6 49 17.42466	+17 56 52.8883	5.49
2015-01-05T12:00:00.00	7 15 7.92468	+17 17 21.3477	5.99
2015-01-06T00:00:00.00	7 40 34.55462	+16 26 1.4659	10.24
2015-01-06T12:00:00.00	8 5 35.78652	+15 23 51.6454	15.35

FIGURE 2.11 – Une partie de la table d'éphémérides lunaires.

### 2.8.1 Interpolation de Bessel

Pour déduire la position d'un corps céleste, avec une certaine précision, à partir d'un table d'éphéméride, il existe une méthode appelée interpolation de Bessel<sup>18</sup>.

**Etapas à suivre (interpolation à quatre valeurs)** Pour déterminer la position  $x$  à un instant  $t$ , il faut :

1. prendre quatre couples date-position  $(d_i, x_i)$  de la table tel que

$$d_{-1} \leq d_0 \leq t \leq d_1 \leq d_2$$

2. calculer les différences premières

$$\begin{aligned} a &= x_0 - x_{-1} \\ b &= x_1 - x_0 \\ c &= x_2 - x_1 \end{aligned}$$

3. calculer les différences secondes

$$\begin{aligned} d &= b - a \\ e &= c - b \end{aligned}$$

4. calculer le facteur d'interpolation

$$n = \frac{t - d_0}{d_1 - d_0}$$

5. utiliser la formule de Bessel

$$x = x_0 + n.b - \left[ \frac{n(1-n)}{4} \right].(d + e)$$

18. cf. bibliographie [10]



### 2.8.2 Précision de l'interpolation de Bessel

Par rapport à une interpolation linéaire, la précision apportée par cette méthode dépendra des valeurs des différences secondes.

Dans le cas où ces dernières seraient très petites, cela indiquerait que la position varie presque linéairement entre les dates concernées.

Dans le cas contraire, la précision du résultat de Bessel sera vérifiée par la différence ternaire.

Si la valeur de cette dernière est trop élevée, il conviendrait de réduire le pas de la table des éphémérides ou de choisir une autre méthode.

## 2.9 Elaboration de la table "constellation"

Une constellation, est un ensemble d'étoiles qui appartiennent à une même région de la voûte céleste.

Dans la plupart des cas, le dessin attribué à une constellation relie uniquement les étoiles faisant partie d'elle.

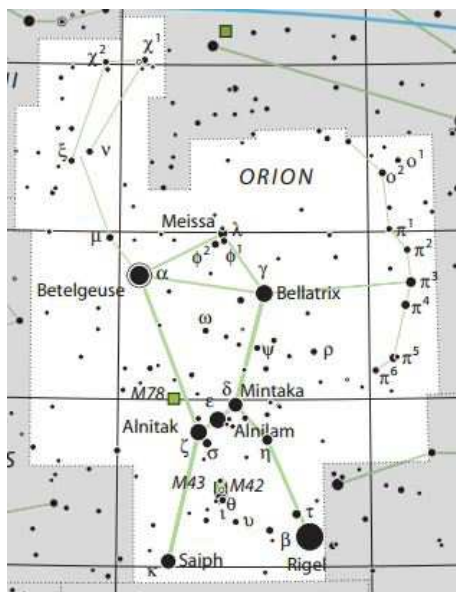


FIGURE 2.12 – La constellation d' Orion (Ori).

**Image :** cf. bibliographie [3].

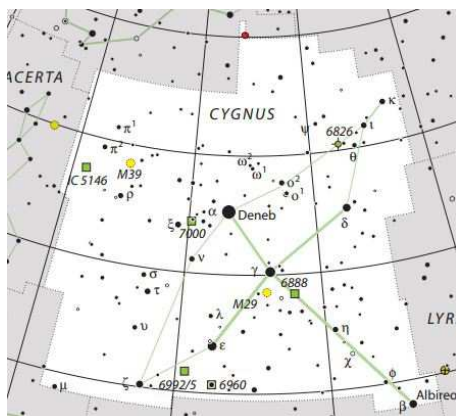


FIGURE 2.13 – La constellation du cygne (Cyg).

**Image :** cf. bibliographie [3].

Nous pouvons déduire de ces dessins, que l'image d'une constellation est constituée d'une ou plusieurs branches, chacune décrivant une suite particulière d'étoiles.

En ce qui concerne la constellation du cygne "Cyg", nous avons :

name	stars_sequence	branch_index
Cyg	zet,nu,alf,iot2,kap,iot2,del,gam,alf,gam,eps,zet,gam,eta,bet1	0,5,9,12
Leo	bet,tet,iot,sig,tet,del,tet,eta,alf,eta,eps,lam,kap,mu,eps,mu,zet,gam1,eta,gam1,del,bet	0,4,6,9,15,19
Ori	nu,chi1,chi2,xi,mu,alf,lam,gam,alf,zet,eps,del,zet,kap,del,eta,bet,del,gam,pi3,pi4,pi5,pi6,pi3,pi2,pi1,omi2	0,12,14,17,23
Lib	tau,ups,gam,alf2,gam,bet,alf2,sig	0,4
Lyr	gam,del2,zet1,alf,zet1,bet,gam	0,4
Peg	gam,alf,bet,eta,pi2,bet,mu,lam,iot,kap,alf,zet,tet,eps	0,5,10
Per	lam,mu,del,eps,xi,zet,omi,eps,bet,ro,bet,kap,iot,tet,phi,iot,alf,iot,tau,eta,gam,tau,gam,alf,del	0,7,10,15,17,22
Psc	phi,tau,ups,phi,eta,omi,alf,nu,mu,eps,del,omg,iot,tet,gam,kap,lam,iot	0
Sge	gam,del,alf,del,bet	0,3

FIGURE 2.14 – Une partie de la table "constellation".

Cela a été déduit par correspondance aux données disponibles au sein de la table stars.

Par cette méthode, nous obtenons au final ce rendu :

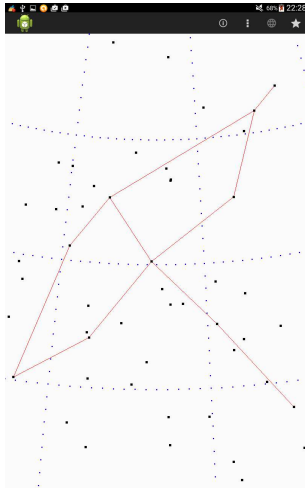


FIGURE 2.15 – Le rendu de la constellation du cygne.

## 2.10 Les types de données à virgule flottante

Les types décrits ci-dessous sont définis par la norme IEEE 754<sup>19</sup>.

(Provenance du tableau<sup>20</sup>.)

type	Encodage	Signe	Exposant	Mantisse	Valeur	Précision	Chiffres significatifs
float	32 bits	1 bit	8 bits	23 bits	$(-1)^S \times M \times 2^{E-127}$	24 bits	environ 7
double	64 bits	1 bit	11 bits	52 bits	$(-1)^S \times M \times 2^{E-1023}$	53 bits	environ 16

### 2.10.1 Conversion du type double en type float

Pour une question de précision, le type double sera utilisé pour l'ensemble des calculs mathématiques.

Malheureusement, pour que les données puissent être traitées par le pipeline de rendu OpenGL, il faut qu'elles soient transmises sous la forme d'un buffer constitué de float car OpenGL ES 2.0 n'admet pas la double précision !

Mais, une conversion de double en float peut entraîner, au delà de la simple perte de précision, les problèmes suivants :

**Infini** : la conversion du double a entraîné une valeur infinie.

**NaN** : la conversion du double n'a pas pu être interprétée comme un nombre.

Mais aussi, de nombreux autres problèmes détaillés sur le site d'Oracle<sup>21</sup>.

Pour palier les problèmes explicités, une méthode sera créée pour assurer cette conversion<sup>22</sup>.

## 2.11 Liaison avec la base de données

Il a été décidé d'inclure la base de données dans un fichier appelé "asset" créé à cet effet.

Dès lors, pour garantir l'accès au contenu de ce fichier, nous faisons hériter les classes "dao" de la classe "SQLiteAssetHelper".

Cependant, "SQLiteAssetHelper" faisant partie d'une librairie externe écrite par l'entreprise "Ready State Software", l'archive java (JAR) disponible sur le site MVNrepository<sup>23</sup> sous la licence Apache 2.0<sup>24</sup> a été préalablement importée dans le dossier "External Libraries".

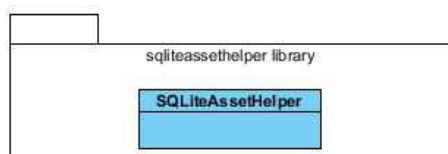


FIGURE 2.16 – SQLiteAssetHelper

19. cf. <http://grouper.ieee.org/groups/754/>. 20. cf. [http://fr.wikipedia.org/wiki/Virgule\\_flottante](http://fr.wikipedia.org/wiki/Virgule_flottante). 21. cf. chapitre 5.1.3 Narrowing Primitive Conversion et 4.2.4 Floating-Point Operations [16] 22. cf. chapitre 3.5. 23. cf. site <http://mvnrepository.com/artifact/com.readystatesoftware.sqliteasset/sqliteassethelper/2.0.1> 24. cf. site <http://www.apache.org/licenses/LICENSE-2.0.txt>

# Chapitre 3

## Construction

### 3.1 Phase d'initialisation

Cette phase doit servir à bâtir le modèle qui initialisera les classes responsables du rendu.

A cet effet, le contrôleur de lancement passera du thread <sup>1</sup> principal, utilisé pour mettre à jour une barre d'avancement du processus, à un thread secondaire responsable de la construction du modèle.

Afin de faciliter ce rôle, "LaunchingController" héritera de la classe AsyncTask.

#### 3.1.1 Diagramme d'interactions entre le modèle et l'activité de lancement

##### Description du diagramme

**Précondition :** La classe "LaunchingController" a été instanciée.

Il existe trois types de lancement :

1. sans position initialisée ;
2. avec position initialisée ;
3. avec attente de la position GPS : son constructeur a alors préalablement instancié la classe "Location\_hac" et lancé son fil d'exécution d'écoute pour la mise à jour de la position par la puce GPS ;

En cas de lancement de type 1 et 2, nous passerons directement au message trois du diagramme .

Dans le cas 3, nous nous placerons en boucle d'attente de la mise-à-jour de la position.

Si cette mise à jour se produit, nous passerons au message numéro trois du diagramme, sinon nous revenons au début du cas d'utilisation "Initialisé la position" <sup>2</sup>.

##### Description des messages :

###### 3 :

Le constructeur de "HeavenCanopy"

1. crée un objet de la classe "RenderManager" ;
2. lance la procédure de création du modèle (3.1.y) .

###### 3.2 :

l'instance de la classe "Ephemeris" (x(4)) transmet le tableau composé des positions et celui composés des couleurs de la Lune et du Soleil, au constructeur de la classe "SolarSystemBodyRender" pour qu'il puisse l'associer à ses buffers de rendu.

###### 3.3 :

"HeavenCanopyController" sera implémentée selon le pattern singleton et possédera une instance de "RenderManager", et une de "HeavenCanopy" (transmises lors de l'appel à son constructeur par "LaunchingController").

Par ce procédé, nous garantissons l'existence de ces trois classes fondamentales, tout au long de l'exécution du programme.

###### 5.2 :

l'activité de lancement passe le fil d'exécution principal à l'activité "HeavenCanopyActivity" et se ferme.

---

1. cf. définition en annexe G.    2. voir figure 1.3.

**d :**

en cas d'erreur, le programme prend fin grâce à l'appel de la méthode "finsh()".

### **Procédure de création du modèle :**

Dans le diagramme ci-dessous, pour chaque y, X(y) est une classe du modèle dont le type sera énuméré selon un indice y qui démarre à zéro en cas de position initialisé où à 1 dans le cas contraire.

Les valeurs "y" correspondent aux classes suivantes :

0. GeoElements;
1. StarsDome;
2. GridRA\_DE;
3. Names;
4. Ephemeris;
5. Moon;
6. Sun;

Le constructeur de chacune de ces classes effectuant les opérations suivantes :

1. crée, récupère et initialise les éléments nécessaires pour assumer ses responsabilités;
2. instancie sa classe soeur de type "render" en lui transmettant lesdits éléments pour un futur rendu graphique (cette classe soeur en profitera pour s'enregistrer auprès de RenderManager).

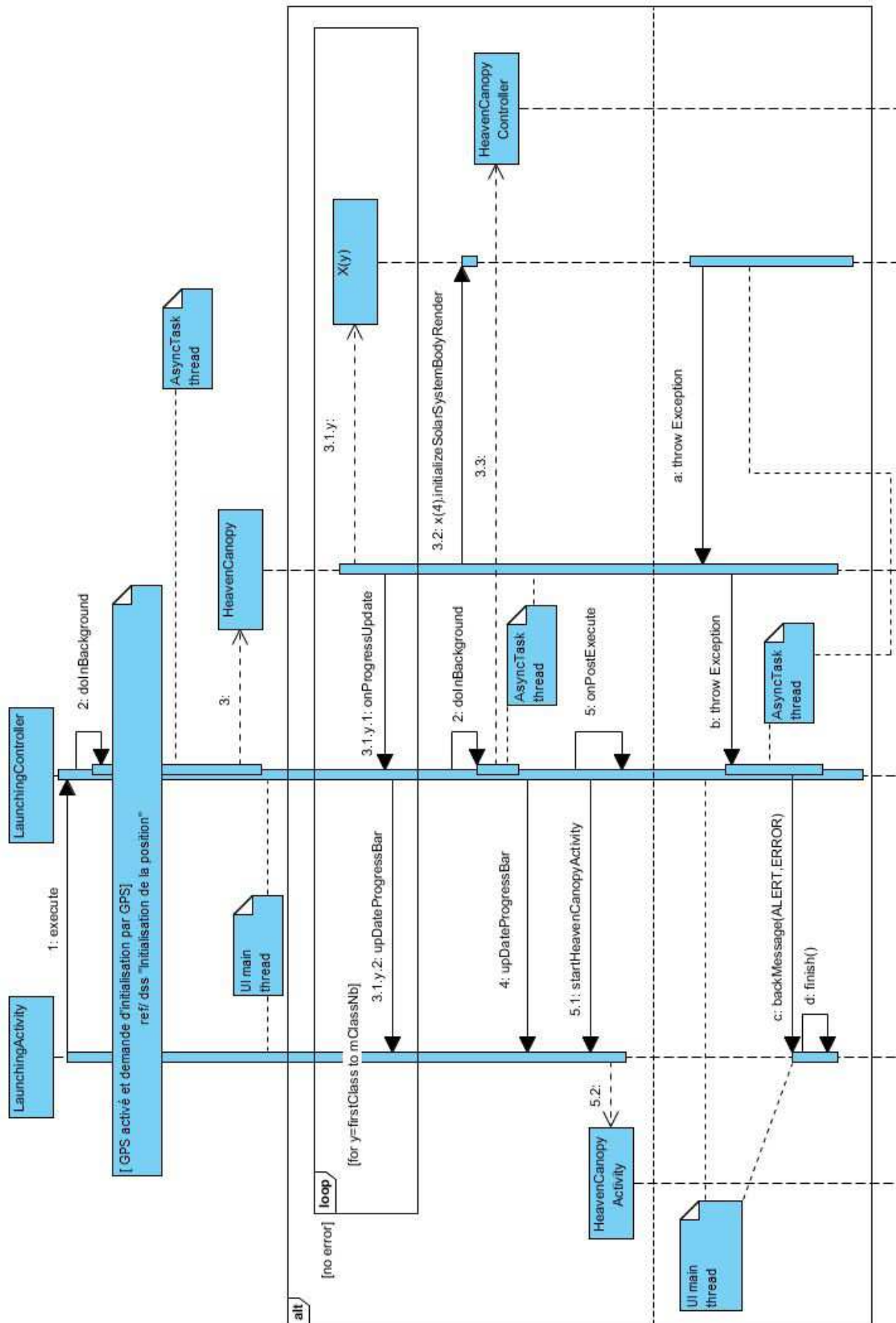


FIGURE 3.1 – La phase d'initialisation par le fil d'exécution AsyncTask.

**Légende** => X(y) est une classe spécifique du modèle. | x(4) correspond à un objet de la classe Ephemeris. | firstClass : indice de la première classe (0 ou 1). | mClassNb : indice représentant la dernière classe.

### 3.1.2 Construction de la classe StarsDome

**Précondition :** lancement du message 3.1.1 de la phase d'initialisation.

**Constructeur :** Stars(Context context, RenderManager renderManager)

**Postcondition :** lancement du message 3.1.1.1 de la phase d'initialisation.

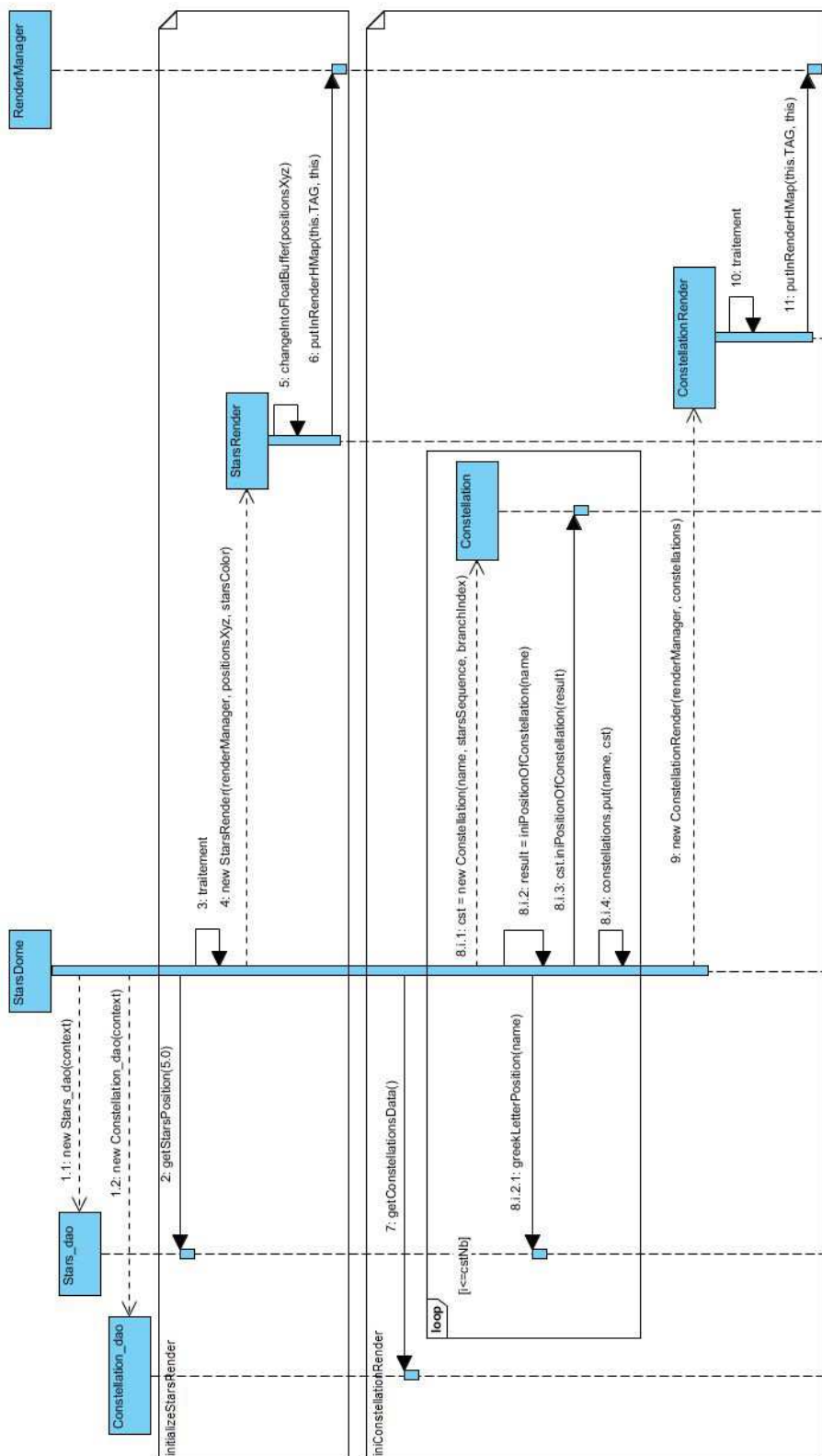


FIGURE 3.2 – Processus de construction de la classe StarsDome.

**Légende** ==> cstNb : nombre d'entrée de la table constellation. | i : itération en cours.



## Description du diagramme

**1 :** context représente la classe Context<sup>3</sup>.

**2 :**  
récupère la position de l'ensemble des étoiles de magnitude  $\leq 5.0$  (l'ensemble de la table stars), sous la forme d'une liste de tableau de type double.

**3 :**  
— transforme les données d'ascension droite-déclinaison en coordonnées xyz de la surface de la sphère unité;  
— convertit les données en type float pour obtenir le tableau de float positionsXyz.

**5 :**  
place les données positionsXyz dans un buffer pour une future utilisation par le pipeline de rendu OpenGL.

**6 et 11 :**  
ajoute l'instance de la classe de rendu à la liste des objets graphiques à gérer.

**7 :**  
récupère la liste des informations concernant l'ensemble des constellation contenue dans la table constellation.

**En boucle** de  $i = 1$  au nombre de constellations traitées par la base de données.

**8.i.1 :**  
instancie une constellation avec les informations lui étant propres obtenue en 7 .

**8.i.2 :**  
— récupère l'ensemble des couples lettre grecque-position des étoiles pour la constellation nommée;  
— opère au même traitement que pour le message 3 sur lesdites positions;  
— result = la liste des couples lettre grecque - position avec la position de type tableau de coordonnées xyz de type float.

**8.i.3 :**  
initialise le tableau de type float représentant la suite de position à suivre pour dessiner la constellation.

**8.i.4 :**  
ajoute l'instance de la constellation à la liste d'instances de constellations appelée "constellations".  
**Fin de boucle.**

**9 :**  
instancie la classe "ConstellationRender" en lui passant comme paramètre la liste des constellations.

**10 :**  
crée une liste de buffers, chaque buffer contenant la suite constituée en 8.i.3 pour la constellation considérée.  
Cette liste sera utilisée par la méthode "draw" de ConstellationRender pour ensuite être interprétée sous la forme d'une suite de primitifs de type GL\_LINE\_STRIP (segment) par le pipeline de rendu OpenGL.

---

3. voir annexe D.3

### 3.1.3 Initialisation de la position d'un corps par interpolation des éphémérides

Ce qui sera décrit ci-dessous, concernera la Lune (la démarche est identique pour le Soleil).

#### Description du diagramme

(cf. "Utilisations des éphémérides" pour plus de précisions<sup>4</sup>.)

- 1 :**  
ajoute sa couleur à la liste de couleurs des corps (du système solaire) de la classe "Ephemeris".
- 2 :**  
initialise sa variable d'instance de position par appel de la méthode d'interpolation de Bessel..
- 2.1 :**  
"now" représente la date et l'heure "UTC"<sup>5</sup> courante.
- 2.2 :**  
fournit la première date de Bessel  $d_{-1}$ .
- 2.3**  
fournit les quatre couples date-position  $(d_i, x_i)$  de Bessel.
- 2.4 :**  
calcule le facteur d'interpolation  $n$ .
- \*2.5 :**  
calcule pour  $i =$  ascension-droite et  $i =$  déclinaison, le résultat de l'interpolation (cf. Implémentation de la méthode "getBesselResult")
- 2.6 :**  
ajoute le couple de résultats à la liste de positions des corps (du système solaire) de la classe "Ephemeris".

---

4. voir chapitre 2.8.    5. cf. définition en annexe G.

## Diagramme d'interactions

**Précondition** : lancement du message 3.1.5 de la phase d'initialisation.

**Constructeur** : Moon(Ephemeris ephemeris)

**Postcondition** : lancement du message 3.1.5.1 de la phase d'initialisation.

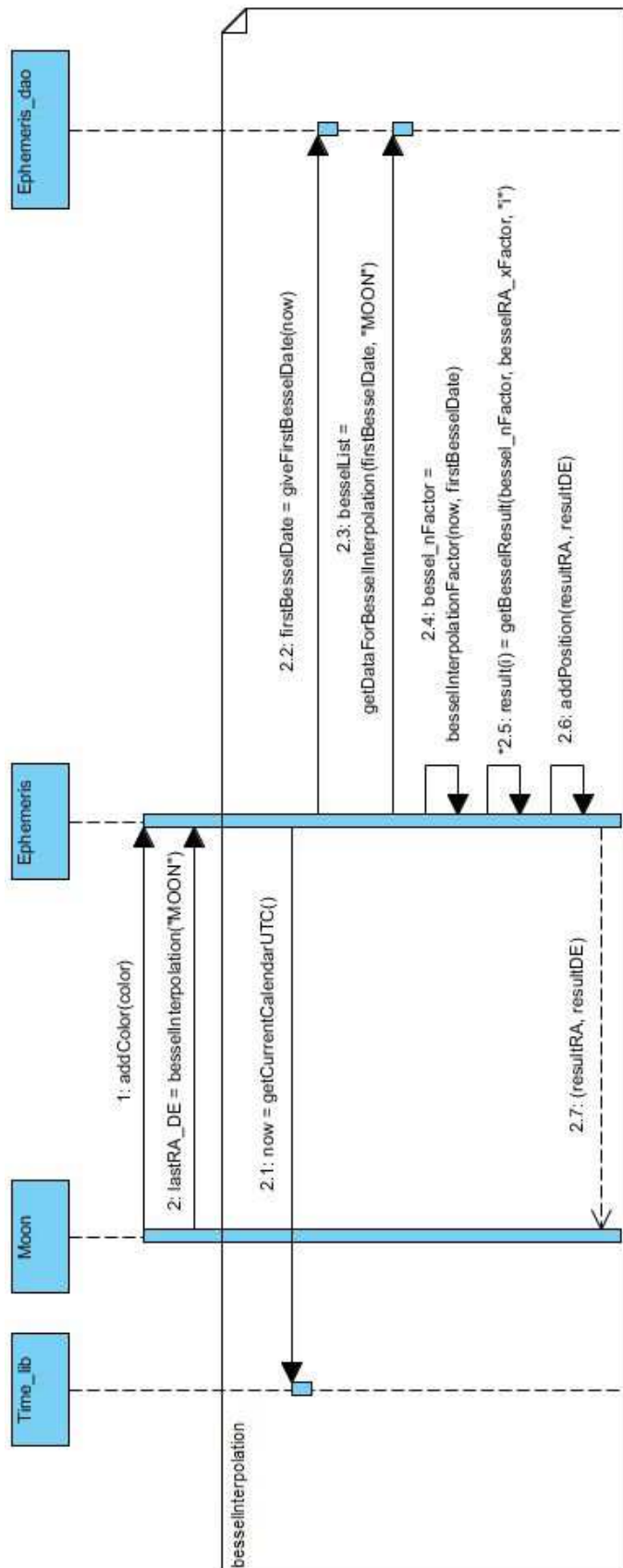


FIGURE 3.3 – Processus de construction de la classe Moon.

Légende => i = RA ou DE.

## Implémentation de la méthode "getBesselResult"

Cette méthode fournit le résultat de l'interpolation de Bessel.

Nous avons aussi les sous méthodes suivantes :

**"getMin"** : fournit la fluctuation minimale des trois différences premières.

**"setIfBesselRA\_anomaly"** :

détecte et corrige un possible saut de valeur d'ascension-droite (RA).

Par exemple : si l'angle RA passe de 359 à 1 degré, la véritable variation est de 2 degrés au lieu de -358 degrés!

Son paramètre "deltaFluctuation" peut aussi servir à tolérer la variation de la vitesse orbitale du corps qui peut différer d'un pas d'éphéméride à l'autre surtout si les pas sont grands.

Ce paramètre est initialisé à 10 degrés par défaut car le calcul précis de celui-ci s'avère complexe.

## La méthode "getBesselResult"

```
private double getBesselResult(double n, double[] bessel_xFactor,
                               String type) throws Exception {
    if (bessel_xFactor.length == 4) {
        double xZero = bessel_xFactor[1];
        double a = bessel_xFactor[1] - bessel_xFactor[0];
        double b = bessel_xFactor[2] - bessel_xFactor[1];
        double c = bessel_xFactor[3] - bessel_xFactor[2];

        if (type.compareTo("RA") == 0) {
            double min = Math_lib.getMin(a, b, c);
            a = setIfBesselRA_anomaly(a, min, 10.0);
            b = setIfBesselRA_anomaly(b, min, 10.0);
            c = setIfBesselRA_anomaly(c, min, 10.0);
        }

        double d = b - a;
        double e = c - b;

        return xZero + n * b - (n * (1 - n) / 4.0) * (d + e);
    } else
        throw new Exception(" bessel_xFactor.length != 4 !!!");
}
```

## La méthode "setIfBesselRA\_anomaly"

```
private double setIfBesselRA_anomaly(
    double value, double min, double deltaFluctuation) {
    if (Math.abs(value) > (min + deltaFluctuation)) {
        if (value > 0) {
            value = value - 360;
        } else { // NEVER 0 FOR ephemeris PAS < 0 !
            value = value + 360;
        }
    }
    return value;
}
```

## Legende :

a,b, et c : sont les différences premières de Bessel.

d et e : les différences secondes.

n : le facteur d'interpolation.

bessel\_xFactor : la liste des valeurs de position (ascension-droite ou déclinaison) fournies par les dates de Bessel.

## 3.2 Phase principale

### 3.2.1 Diagramme d'initialisation de la phase principale

**Précondition :** lancement du message 5.2 de la phase d'initialisation.

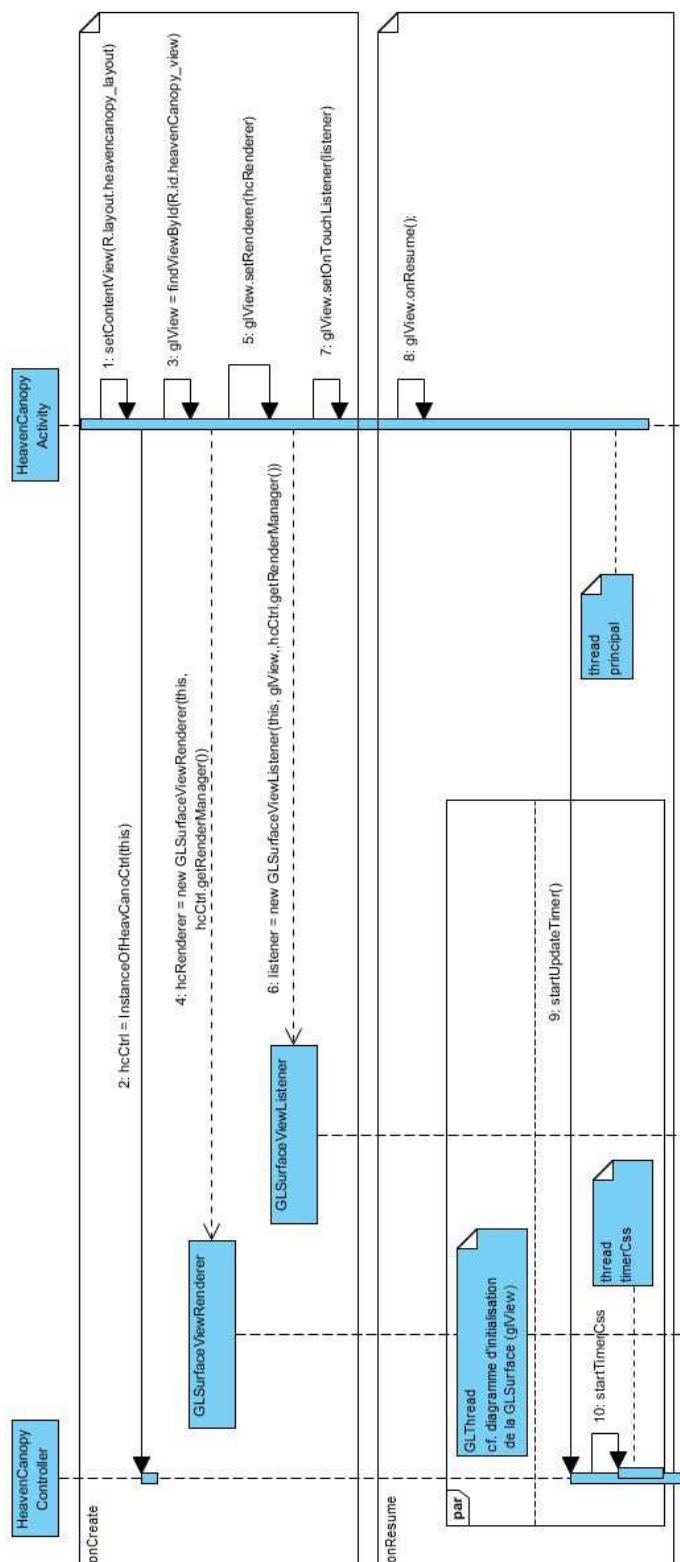


FIGURE 3.4 – Initialisation de la séquence principale.

**Légende** => onCreate et onResume sont deux méthodes faisant parties du cycle de vie de l'activité.

## Description du diagramme

- 1 :**  
l'activité se lie à son implémentation graphique (xml).
- 2 :**  
récupère l'instance du contrôler principal.
- 3 :**  
se lie à la surface OpenGL déclarée dans son fichier d'implémentation graphique (xml).
- 4 et 5 :**  
instancie la classe responsable de la gestion des événements propres à une surface OpenGL et la lie à ladite surface.
- 6 et 7 :**  
instancie un écouteur de surface et le lie à la surface OpenGL.
- 8 :** appel à la méthode "onResume()" de la surface OpenGL.

### Parallèlement :

- la méthode onResume() de la GLSurfaceView, ajoute sur la pile d'exécution du GLthread les appels aux méthodes "onSurfaceCreated", "onSurfaceChanged" et "onDrawFrame" de la classe "GLSurfaceViewRenderer"  
(cf. diagramme d'initialisation de la GLSurface) ;
  - le fil d'exécution principal passe au message 9.
- 9 et 10 :**  
demande et lancement du fil d'exécution des mises à jour de la position de la Lune et du Soleil (thread timerCss).

### 3.2.2 Diagramme d'initialisation de la GLSurface

**Précondition :** appel de la méthode onResume() de l'OpenGL surface.

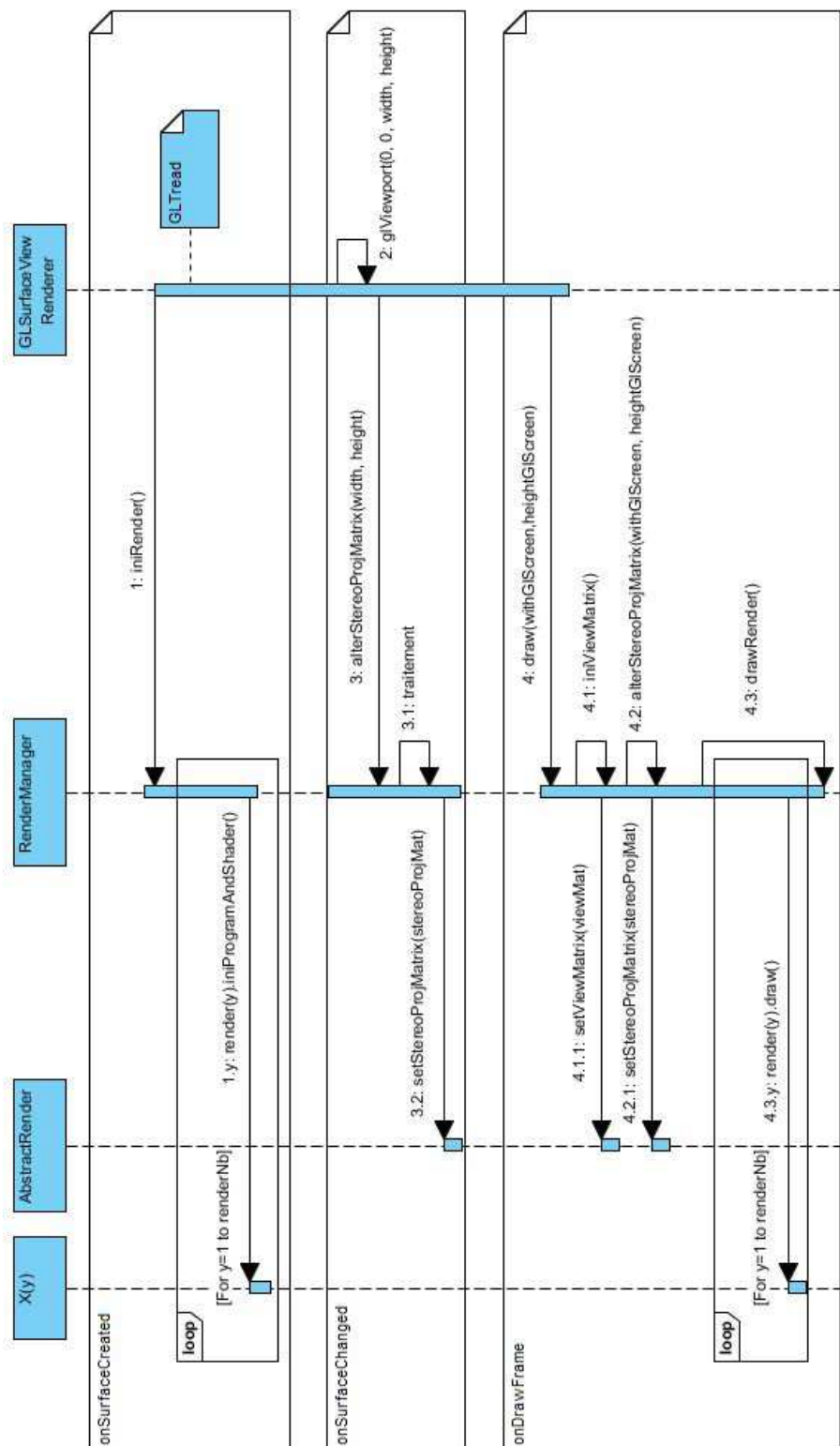


FIGURE 3.5 – Initialisation de la surface de rendu OpenGL.

**Légende** => renderNb : nombre de classes de rendu. | render(y) : une instance d'une classe de rendu spécifique.

## Description du diagramme

Les appels aux méthodes "onSurfaceCreated", "onSurfaceChanged" et "onDrawFrame" de la classe "GLSurfaceViewRenderer" se succèdent dans la pile d'exécution du thread OpenGL.

- 1 :**  
demande à la classe RenderManager d'initialiser le programme de liaison CPU-GPU<sup>6</sup> et les shaders<sup>7</sup> de chacune de ses classes de rendu.
- 2 :**  
définit le rectangle sur lequel le rendu apparaîtra<sup>8</sup>.
- 3 et 3.1 :**  
demande à la classe RenderManager d'initialiser la matrice responsable de la projection.
- 3.2 :**  
remplace la matrice de projection utilisée par l'ensemble des classes de rendu.
- 4 :**  
demande à la classe RenderManager de lancer la procédure de rendu graphique.
  - 4.1 et 4.1.1 :**  
initialise et remplace la matrice de vue utilisée par l'ensemble des classes de rendu.
  - 4.2 et 4.2.1 :**  
met à jour et remplace la matrice de projection utilisée par l'ensemble des classes de rendu.
  - 4.3 :**  
la classe RenderManager appelle la méthode "draw()" de chacune des classes de rendu activées. Dans chacune de ces classes, cette méthode lance le programme de liaison CPU-GPU afin d'effectuer le rendu graphique.

---

6. cf. définition en annexe G.    7. voir annexe E.4.1 et E.4.7. Définition en annexe G.    8. voir annexe E.5.6.



### 3.2.3 Initialisation du repère azimutal

#### 3.2.3.1 Diagramme d'initialisation

**Précondition :** l'utilisateur a initialisé sa position et a sélectionné le mode azimutal.

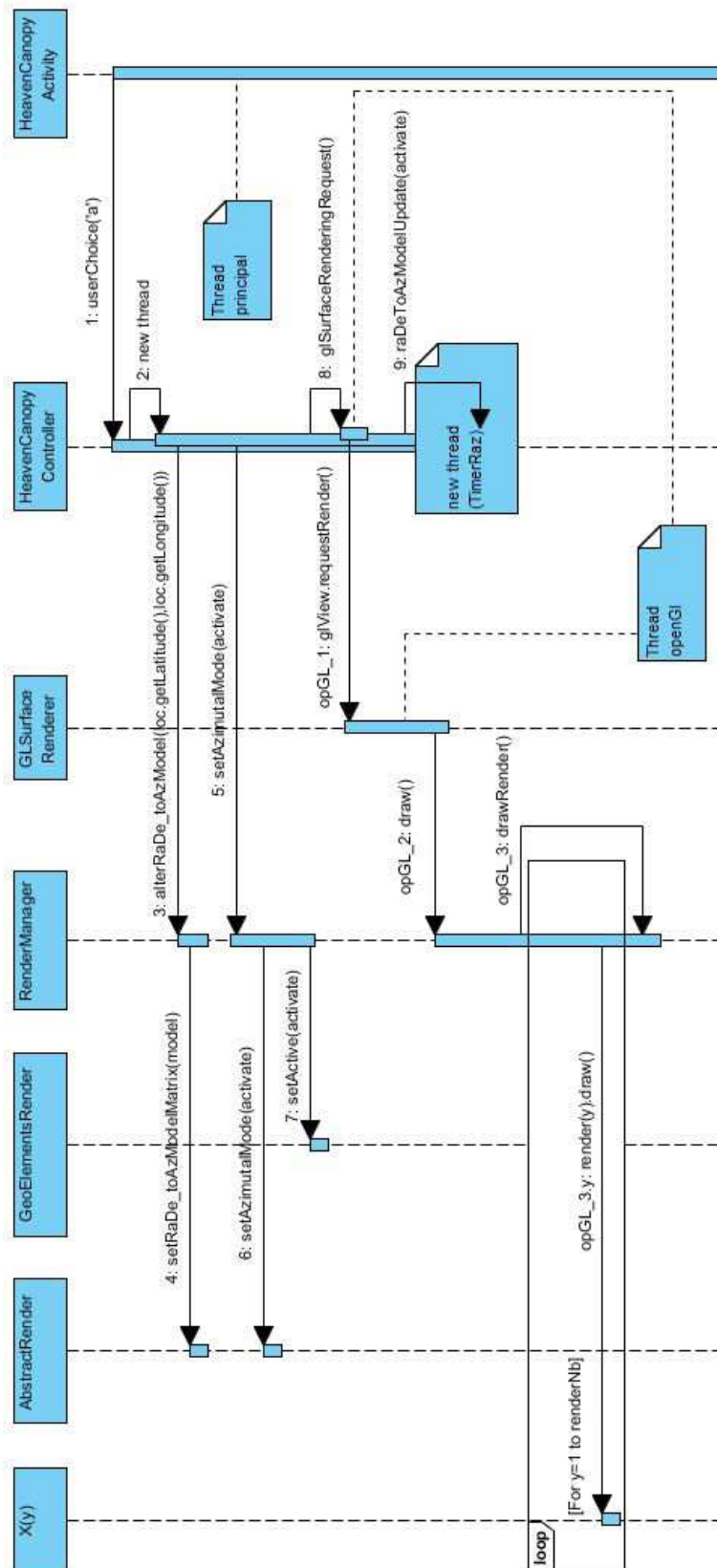


FIGURE 3.6 – Activation du repère azimutal.

Légende => "a" signifie repère azimutal

## Description du diagramme

- 1 : l'utilisateur sélectionne le repère azimutal.
- 2 : "HeavenCanopyController" lance le fil d'exécution de l'initialisation du repère azimutal.
- 3 :  
demande au RenderManager d'initialiser la matrice de transformation de repère à l'aide des paramètres de position récupérées dans la classe Location\_hac.
- 4 :  
avec cette dernière, initialise ou remplace la matrice spécifique du modèle, utilisée à cet fin, par l'ensemble des classes de rendu étant de type équatorial.
- 5 :  
demande d'activation du mode azimutal.
- 6 :  
active le mode azimutal de rendu.
- 7 :  
active la classe responsable du rendu des éléments géographiques (horizon et méridien) pour des futures demandes de rendu.
- 8 :  
le contrôleur lance sa procédure de rendu (thread OpenGL).  
**opGL\_1 :**  
ajoute une requête de rendu de la surface OpenGL sur la pile d'exécution du thread OpenGL.  
**opGL\_2 et opGL\_3 :**  
La classe RenderManager reçoit une demande de rendu et appelle la méthode "draw()" de chacune des classes de rendu activées.  
Dans chacune de ces classes, la méthode "draw()" initialise et lance le programme de liaison CPU-GPU pour effectuer le rendu graphique.
- 9 :  
procède au lancement du fil d'exécution des mises à jour de la mise à jour du repère azimutal (thread timerRaz).

### 3.2.3.2 Implémentation de la matrice de transformation de repère

La méthode ci-dessous permet de transformer les positions établies sous le repère équatorial en des positions azimutales.

Elle se décompose en trois parties :

1. une rotation vers l'est du plan xy de l'équateur d'un angle égal au temps sidéral local (exprimé en degré). Autrement dit, cette transformation place l'axe contenant le point vernal à un temps sidéral local de l'axe x du repère horaire ;
2. une rotation vers l'ouest du plan xy de l'équateur d'un angle égal à 90 degrés. Autrement dit, cette transformation place l'axe contenant le point vernal à un temps sidéral local de l'axe y du repère horaire (axe d'origine des azimuts = méridien de l'observateur) ;
3. une rotation du plan yz du méridien d'un angle complémentaire à la latitude vers le nord. Autrement dit, cette transformation place l'axe contenant les pôles à un angle complémentaire de l'axe azimutal zénith-nadir.

@Override

```
public synchronized void alterRaDe_toAzModel(  
    double latitude, double longitude) throws Exception {  
    float[] model = new float[16];  
    Matrix.setIdentityM(model, 0);  
    //ORIGINE OF VERNAL DOT = axe x of equatorial system !  
    // put axe x of equatorial (vernal dot) at sideral time of Hours system  
    Matrix.multiplyMM(  
        model, 0, Math.lib.get4RotMatrixEquatorialToHoursCoordinates(  
            Time.lib.getLocaleMeanSideralTime(longitude)*15), 0, model, 0);  
    //Change x to y axe for Hours system  
    Matrix.multiplyMM(  
        model, 0, Math.lib.rot4OfRepereXYCounterclockMatrix(90.0), 0, model, 0);  
    //put axe z (polar) on polar axe of azimuth system  
    Matrix.multiplyMM(model, 0,  
        Math.lib.get4RotMatrixHoursToAzimuthCoordinates(90.0 - latitude), 0, model, 0);  
    AbstractRender.setRaDe_toAzModelMatrix(model);  
}
```

### 3.2.4 Construction de la fonctionnalité de rendu de noms

L'algorithme explicité ci-dessous, ajoute à la séquence d'instruction de base (brièvement explicité dans le chapitre consacré à l'élaboration) l'aspect lié à la transmission de texture.

Cette responsabilité additionnelle la rend sensiblement plus compliquée que les autres classes qui se consacrent seulement à la transmission de position et de couleur pour le rendu de primitive de type point (ou de segment pour celle consacrée au rendu des constellations).

Pour plus de précisions concernant les notions abordées dans cette section, il est utile de consulter l'annexe sur OpenGL.

#### 3.2.4.1 Activation de la classe de rendu des noms

##### Diagramme d'activation

**Précondition :** l'utilisateur a touché deux fois, d'une manière rapide, la même zone de l'écran.

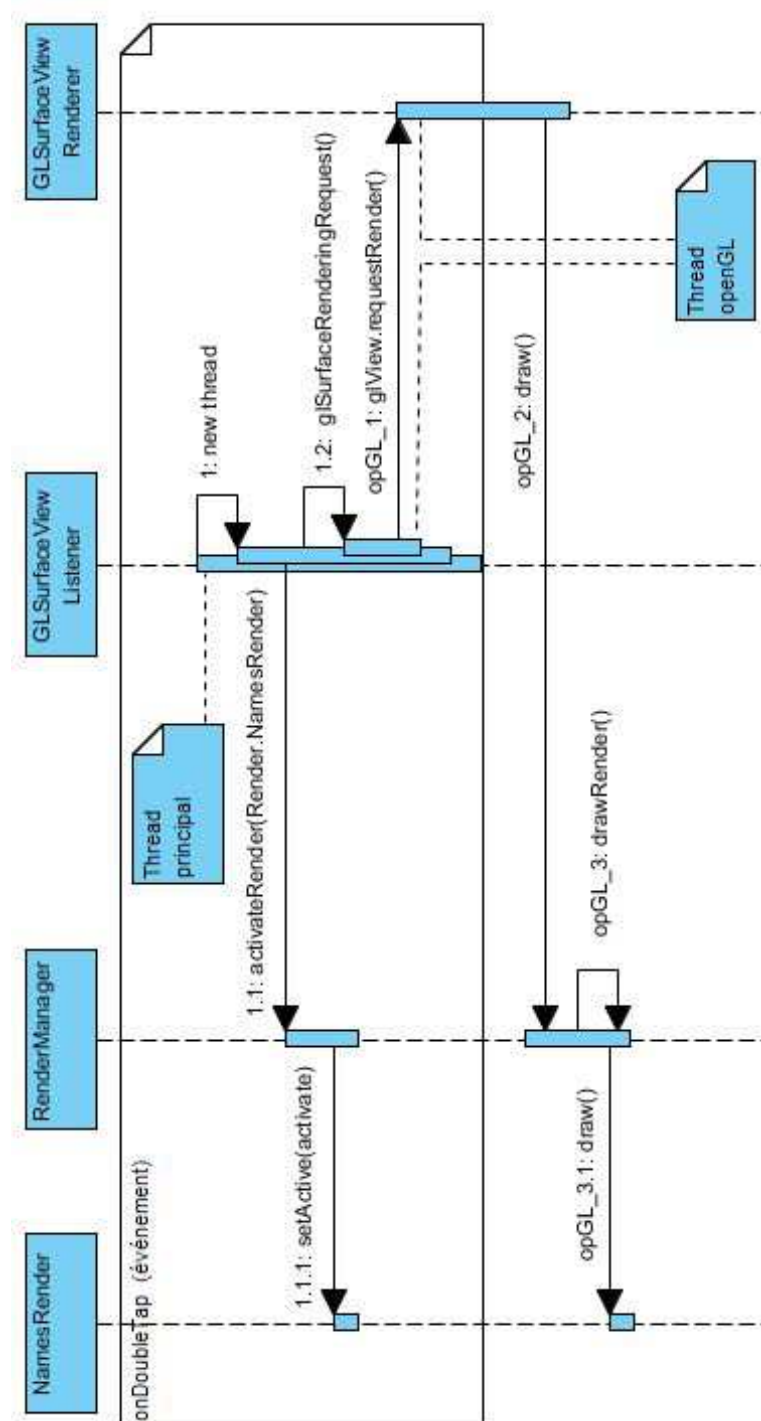


FIGURE 3.7 – Activation de l’affichage des noms.

## Description du diagramme

### **1 :**

l'écouteur de la surface graphique capte l'événement et lance un nouveau fil d'exécution.

### **1.1.1 :**

modifie l'état d'activation de la classe responsable du rendu des noms :

#### **activation :**

recupère les images contenues dans le dossier ressource "drawable-nodpi" et initialise des variables de type collections avec ces dernières

#### **désactivation :**

supprime les variables de type collection initialisées lors de l'activation.

### **opGL\_1 :**

l'écouteur ajoute une requête de rendu de la surface OpenGL sur la pile d'exécution du thread OpenGL.

### **opGL\_2 et opGL\_3 :**

La classe "RenderManager" reçoit une demande de rendu et appelle la méthode "draw()" de chacune des classes de rendu activées.

Dans chacune de ces classes, la méthode "draw()" initialise et lance le programme de liaison CPU-GPU pour effectuer le rendu graphique.

### 3.2.4.2 Implémentation du VertexShader et du FragmentShader

Le code ci-dessous participe au rendu des images.

Les mots clés utilisés ci-dessous étant définis dans l'annexe consacré à OpenGL seul le principe de fonctionnement sera expliqué ici.

#### Eléments utilisés par les "shaders" :

Pour chaque texture à rendre, les shaders parcourront les données qui leur ont été transmises dans l'ordre imposé par un tableau d'indices (0,1,2,3,0)<sup>9</sup> transmis lors de l'appel à la méthode "glDrawElements" (voir l'implémentation ici plus bas).

Les données transmises sont :

- la matrice de transformation pour l'élément "u\_MVPMatrix"
- les coordonnées de texture pour l'élément "a\_texCoord"
- le cadre de position où l'on désire placer la texture pour l'élément "a\_Position"
- l'image pour l'élément "s\_texture"

Les coordonnées de texture :

```
(0,0) -- (1,0)
|         |
(0,1) -- (1,1)
float [] uvs = new float [] {
    0.0f, 1.0f, //indice 0
    1.0f, 1.0f, //indice 1
    1.0f, 0.0f, //indice 2
    0.0f, 0.0f, //indice 3
}
```

#### Le VertexShader

Le programme ci-dessous fonctionne de la manière suivante : pour chaque indice "i" de la suite définie ci-dessous sous le terme "indices", le programme puise dans ses buffers l'ensemble de données correspondant au type vecteur défini pour l'attribut concerné et effectue les opérations suivantes :

1. calcule la position du sommet i (avec sa valeur d'homogénéité) du cadre de position en lui appliquant la matrice de transformation.
2. confie la position i du cadre de coordonnées texture "uvs" au FragmentShader.

```
private final String vertexShader =
    "attribute vec3 a_Position;" +
    "uniform mat4 u_MVPMatrix;" +
    "attribute vec2 a_texCoord;" +
    "varying vec2 v_texCoord;" +

    "void main() {" +
    "    gl_Position = u_MVPMatrix * vec4(a_Position, 1.0);" +
    "    v_texCoord = a_texCoord;" +
    "}";
```

---

9. voir chapitre 2.7.3.

## Le FragmentShader

Le programme ci-dessous accède aux données de l'image contenue dans "s\_texture" en fonction de la position i du cadre de coordonnées texture reçues par le VertexShader et contenue dans "v\_texCoord".

1. La fonction "texture2D" reçoit l'image et les coordonnées de textures et retourne les données récoltées (couleur, transparence,...) sous la forme de texel<sup>10</sup>.
2. "gl\_FragColor" transmet les données récoltées au pipeline graphique pour la suite des opérations de rendu.

```
private final String fragmentShader =
    "precision mediump float;" +
    "varying vec2 v_texCoord;" +
    "uniform sampler2D s_texture;" +

    "void main() {" +
    "    gl_FragColor = texture2D( s_texture , v_texCoord );" +
    "}";
```

### 3.2.4.3 Implémentation des méthodes utilisées pour le rendu des noms

La méthode "draw()" commence en premier lieu par récupérer la matrice de transformation composée du produit de la matrice du modèle avec celle de la vue et de la projection.

Cependant, étant donné que la position des noms est liée au repère équatorial, la matrice du modèle utilisé sera celle de transformation de repère dans le cas où le mode azimutal serait préalablement activé.

En deuxième lieu, la méthode "draw()" fait appel à deux sous méthodes équivalentes, "drawConstellationsName()" pour le rendu de nom de constellation et à "drawStarsName()" en ce qui concerne les noms d'étoiles.

#### drawStarsName()

Cette méthode permet sous l'exécution d'un programme, de transmettre au pipeline graphique du GPU les données (positions, images,...) et au GPU d'exécuter les programmes "Vertex Shader" et "Fragment Shader".

Dans la suite de la section, nous verrons apparaître un certain nombre de fois, la méthode "catchErrorOpenGL".

Cette méthode est utile pour attraper les erreurs liées aux traitements spécifiques qu'effectuent les bibliothèques d'OpenGL<sup>11</sup>.

En ce qui concerne "drawStarsName", son algorithme se décompose en huit parties :

1. initialise le programme chargé de la liaison entre le GPU et le CPU en y liant les programmes "Vertex Shader" et "Fragment Shader" préalablement initialisés avec leur code d'implémentation et compilés ;
2. "glUseProgram" exécute ledit programme ;
3. attribue la matrice de transformation à la variable u\_MVPMatrix du VertexShader ;
4. attribue le buffer composé des positions "uvs" à la variable a\_texCoord du VertexShader ;
5. prépare la variable s\_texture du VertexShader à recevoir l'image ;
6. fait appel à la méthode drawTextures(...) en lui transmettant la collection de couples, chacun composé d'un bitmap et de, sous la forme d'un buffer, son cadre de position ;
7. désactive toutes les attributions ;
8. et enfin, désactive et détruit le programme et ses shaders.

```
private void drawStarsName() throws Exception{
    if(programHandle== -1)
        iniProgramAndShader(vertexShader, fragmentShader);
    GLES20.glUseProgram(programHandle);

    int matrixMVP_GlslLocation_b = prepareMatrixMVPForShader();
    catchErrorOpenGL("drawStarsName", 1,
```

10. cf. définition en annexe G. 11. cf. chapitre 4.1.2.

```

        "matrixMVP_GlslLocation_b", matrixMVP_GlslLocation_b, false);
int texture_GlslLocation_b = prepareTextureFrameForShader();
catchErrorOpenGL("drawStarsName", 2,
        "texture_GlslLocation_b", texture_GlslLocation_b, false);
int textureHandle_GlslLocation_b = prepareTextureHandleForShader();
catchErrorOpenGL("drawStarsName", 3,
        "textureHandle_GlslLocation_b", textureHandle_GlslLocation_b, false);

drawTextures(stars_texturesBitmapAndBuffer);
// Disable vertex array
GL20.glDisableVertexAttribArray(textureHandle_GlslLocation_b);
GL20.glDisableVertexAttribArray(texture_GlslLocation_b);
GL20.glDisableVertexAttribArray(matrixMVP_GlslLocation_b);
// Desactiver
GL20.glUseProgram(0);
// delete :
deleteProgramAndShader();
}

```

### drawTextures(...)

La méthode ci-dessous, pour chaque texture, se décompose en les étapes suivantes :

1. attribue un identifiant à la texture ;
2. transmet à l'élément "a\_Position" du VertexShader le cadre de position dans lequel on veut rendre l'image ;
3. setupImage associe l'image à l'élément "s\_texture" ;
4. glDrawElements indique quel type de primitives à rendre, le nombre d'éléments à rendre, le type d'élément et les indices où ils sont enregistrés.
5. délée la texture et désactive l'attribut "a\_Position" du VertexShader

```

private void drawTextures(
    HashMap<Bitmap, FloatBuffer> texturesBuffer) throws Exception{
    Set<Bitmap> keySet = texturesBuffer.keySet();
    int glTexture_i;
    int i=0;
    for (Bitmap bmp : keySet) {
        glTexture_i = GL20.GL_TEXTURE + i;
        int position_GlslLocation =
            preparePositionForShaderTxt(texturesBuffer.get(bmp));
        catchErrorOpenGL("drawTextures", 1,
            "position_GlslLocation", position_GlslLocation, false);
        setupImage(bmp, glTexture_i);
        GL20.glDrawElements(GL20.GL_TRIANGLES, indices.length,
            GL20.GL_UNSIGNED_SHORT, drawListBuffer);
        catchErrorOpenGL("drawTextures", 2,
            "position_GlslLocation", position_GlslLocation, false);
        // Bind to default texture
        GL20.glBindTexture(GL20.GL_TEXTURE_2D, 0);
        // Disable vertex array
        GL20.glDisableVertexAttribArray(position_GlslLocation);
        ++i;
    }
}

```



### 3.3 Implémentation des shaders de la classe StarsRender

L'implémentation ci-dessous, est à quelques nuances près identique à celle des autres classes de rendu (excepté celle du rendu des noms).

#### Le VertexShader

**Contenu des variables du shader :**

- "u\_MVPMatrix" contient la matrice de transformation (matrice modèle-vue-projection);
- "a\_Position" contient la position d'une étoile.

**Opérations :**

1. calcule la nouvelle position en appliquant la matrice de transformation sur position fourni par "a\_Position" (auquel on a préalablement ajouté sa valeur d'homogénéité).
2. attribue un pixel de taille 3.0 à cette primitive (dans le cas d'une étoile la primitive est de type point).

```
private final String vertexShader =  
    "attribute vec3 a_Position;" +  
    "uniform mat4 u_MVPMatrix;" +  
  
    "void main() {" +  
        " gl_Position = u_MVPMatrix * vec4(a_Position,1.0);" +  
        " gl_PointSize = 3.0;" +  
    "}";
```

Les données contenues dans "gl\_Position" et "gl\_PointSize" sont ensuite fournies au pipeline pour les étapes suivantes de rendu.

#### Le FragmentShader

**Contenu des variables du shader :**

- "color" contient les données correspondant à la couleur et à la transparence de l'étoile;

**Opérations :** "gl\_FragColor" récupère la couleur et la transparence contenue dans la variable "color".

```
private final String fragmentShader =  
    "precision mediump float;" +  
    "uniform vec4 color;" +  
  
    "void main() {" +  
        " gl_FragColor = color;" +  
    "}";
```

Les données contenues dans "gl\_FragColor" seront ensuite fournies, pour chaque pixel, au pipeline pour les étapes suivantes de rendu.

### 3.4 Implémentation de la méthode de changement de point de vue

La méthode ci-dessous permet d'obtenir à partir d'un mouvement sur l'écran, la nouvelle direction pour l'axe de vue.

Elle se décompose en quatre parties :

1. recherche des coordonnées de type écran du centre de vue existant avant le mouvement ;
2. recherche de la nouvelle direction pour l'axe de vue correspondant au nouveau point sur l'écran (voir la méthode "getWorldXyzAimPoint" située ici plus bas).
3. empêche l'axe de vue de plonger à l'intérieur des cercles polaires de rayon égal à 0,0396.
4. modifie le point de vue de l'utilisateur (voir la méthode "alterViewData" située ici plus bas).

@Override

```
public synchronized void setCenterOfView(double dx, double dy) {
    double win_dx = dx;
    double win_dy = -dy;
    double [] newViewXyzDirection;
    try {
        float [] winXyzCenter =
            Math.lib.obtainWinXyz(viewXyzDirection, getModelView(),
                AbstractRender.getStereoProjMatrix(), getViewport());
        newViewXyzDirection = getWorldXyzAimPoint(
            winXyzCenter[0] + win_dx, winXyzCenter[1] + win_dy);
        //Area authorized for ViewXyzDirection[2] = [-0.98,0.98]:
        newViewXyzDirection[2] =
            Math.max(Math.min(newViewXyzDirection[2], 0.98), -0.98);

    } catch (Exception e) {
        Log.e(TAG + " setCenterOfView ", e.getMessage());
        newViewXyzDirection = null;
    }
    //Area not authorized for radius of polar circle < 0.0396:
    if ((newViewXyzDirection != null) &&
        !isInsidePolarCircle(0.0396, newViewXyzDirection)) {
        alterViewData(newViewXyzDirection);
    }
}
```

#### getWorldXyzAimPoint

La méthode ci-dessous interpole la position sur la voûte à partir d'un point situé sur l'écran.

Elle se décompose trois parties :

1. recherche de la position projetée sur le plan arrière du volume de projection.
2. recherche de la position projetée sur le plan avant du volume de projection.
3. recherche et retourne le point d'intersection entre la droite formée à partir de ces deux dernières positions et l'équation d'une sphère de rayon unité.

@Override

```
public synchronized double [] getWorldXyzAimPoint(
    double winX, double winY) throws Exception {
    float [] modelView = getModelView();
    float [] projectionMatrix = AbstractRender.getStereoProjMatrix();
    int [] viewport = getViewport();
    double [] newWorldXyz_of_newWinXy0 =
        Math.lib.obtainWorldXyz(winX, winY, 0.0,
            modelView, projectionMatrix, viewport);
    double [] newWorldXyz_of_newWinXy1 =
        Math.lib.obtainWorldXyz(winX, winY, 1.0,
            modelView, projectionMatrix, viewport);
    return Math.lib.obtainWorldXyzPerInterpolation(newWorldXyz_of_newWinXy0,
```

```

        newWorldXyz_of_newWinXy1 );
    }

```

## alterViewData

Pour conclure, la méthode ci-dessous met à jour les éléments déterminant la matrice de vue.

Elle se décompose en trois parties :

1. met à jour la position de l'oeil (le point diamétralement opposé au centre de visé).
2. recherche la nouvelle direction du pôle nord et initialise la direction perpendiculaire à l'axe du vue avec cette dernière.
3. met à jour l'axe de vue avec la nouvelle direction obtenue par l'algorithme.

```

private void alterViewData(double [] newViewXyzDirection) {
    try {
        for (int i = 0; i < 3; ++i) {
            eyeXyzPosition[i] = -newViewXyzDirection[i];
        }
        upXyzDirection = Math_lib.viewDirXyz_to_towardOfNorthUpDirXyz(
            newViewXyzDirection, shereRadius);
        this.viewXyzDirection = newViewXyzDirection;
    } catch (Exception e) {
        Log.e(TAG + " alterViewDataXYZ ", e.getMessage());
    }
}

```

## 3.5 Implémentation du contrôle de conversion double-float

### Description

La méthode "convertFromDouble" vérifie si la simple conversion d'ajustement "(float)" ne produirait pas une erreur, un NaN<sup>12</sup> ou un infini<sup>13</sup>.

Si un problème survient, elle appelle "float convertStringOfDouble(double d)" pour obtenir un float convenable ayant au maximum six chiffres significatifs.

Son principe est le suivant :

1. Conversion du double en une chaîne de caractères.
2. Division en 2 ou trois parties :
  - la partie entière;
  - la partie décimale;
  - l'exposant si il existe.
3. Création d'une nouvelle chaîne composée de 6 chiffres significatifs par adaptation, découpage et concaténation de ces parties.
4. Conversion de la nouvelle chaîne en un float.

### Codes

La méthode ci-dessous contrôle l'opération de conversion d'un type de double-précision en un type de simple -précision.

```
public static float convertFromDouble(double d) throws Exception {
    float f;
    try {
        f = (float) d;
        if (Float.isInfinite(f) || Float.isNaN(f))
            f = convertStringOfDouble(d);
    } catch (Exception e) {
        f = convertStringOfDouble(d);
    }
    return f;
}

private static float convertStringOfDouble(double d) throws Exception {
    //TO DO : not consider integer part of 0.12 like a significant letter
    String strReply;
    String str = String.valueOf(d);
    String[] strDotPartition = str.split(Pattern.quote("."));
    if (strDotPartition.length != 2)
        throw new Exception("convertStringOfDouble problem !");
    else {
        String[] strDecimalExpoPartition =
            strDotPartition[1].split(Pattern.quote("E"));
        //DECIMAL PART ALGORITHM
        int integerPartSize = strDotPartition[0].length();
        String appropriateDecimalPart =
            getAppropriateDecimalPart(
                strDecimalExpoPartition[0], integerPartSize);
        //EXPONENT PART ALGORITHM
        String expoPart;
        if (strDecimalExpoPartition.length == 2) {
            expoPart = getAppropriateExpoPart(strDecimalExpoPartition[1], 30);
            //INTEGER PART ALWAYS SIZE=1 IN SCIENTIFIC NOTATION !
            strReply =
                strDotPartition[0].concat(".").
                    concat(appropriateDecimalPart).concat(expoPart);
        } else {
            // CHECK IF INTEGER PART <=6
            if (integerPartSize <= 6) {
```

12. NaN signifie n'est pas un nombre. 13. voir chapitre 2.10.1.

```

        strReply = strDotPartition[0].concat(".").
            concat(appropriateDecimalPart);
    } else {
        //TO DO: round significantIntegerPart and check
        // if expo > 30 (never the case in this app)
        int expo = integerPartSize - 6;
        expoPart = "E".concat(String.valueOf(expo));
        String significantIntegerPart = strDotPartition[0].substring(0,6);
        strReply =
            significantIntegerPart.concat(".").
                concat(appropriateDecimalPart).concat(expoPart);
    }
}
return Float.valueOf(strReply);
}
}

```

# Chapitre 4

## Transition

### 4.1 Test utilisateur

#### 4.1.1 Utilisation du GPS

Lors de l'initialisation de la position par GPS, il arrive que la mise à jour attendue ne se déclenche pas.

La résolution de ce problème, si celui-ci ne serait pas toujours lié à un problème de captation du signal émis par les satellites, fera partie des améliorations futures à apporter à cette application.

#### 4.1.2 Problème lié au rendu des noms

Lorsque l'on sollicite l'affichage des noms, il peut arriver après un certain temps d'utilisation, que l'application décide de fermer la surface de rendu OpenGL et l'activité qui lui est liée.

L'analyse et la résolution dudit problème feront partie des développements futurs.

Néanmoins, la méthode ci-dessous a déjà été construite afin d'obtenir un maximum d'informations sur les erreurs que peuvent déclencher les méthodes spécifiques de la librairie OpenGL.

```
public void catchErrorOpenGL(String method,int methodLocationNb,
    String idName,int idValue,boolean launchException) throws Exception {
    int error = GLES20.glGetError();
    if (error != GLES20.GL_NO_ERROR) {
        Log.e(TAG, method + " " + methodLocationNb + " | ErrorOpenGL " +
            error + " | idName : " + idName + " | idValue : " + idValue);
        if (launchException)
            throw new Exception(TAG + " : " + method + ": glError " + error);
    }
}
```

#### Description des paramètres :

**method** : indique le nom de la fonction dans laquelle elle opère.

**methodLocationNb** : indique la localisation de celle-ci par rapport à des soeurs qui pourraient être présentes au sein de la même fonction.

**idName** : indique le nom de la variable à laquelle elle s'intéresse.

**idValue** : indique la valeur de la variable à laquelle elle s'intéresse.

**launchException** : si elle doit lancer une exception.

## 4.2 Tests unitaires

### 4.2.1 Tests unitaires dao

#### 4.2.1.1 Constellation\_daoTest

Dans les tests ci-dessous, la valeur attendue doit correspondre au nombre de constellations présentes au sein de la table "constellation".

```
@MediumTest
public void testGetConstellationsData() {
    try {
        ArrayList<String[]> result =
            cst_dao.getConstellationsData();
        int dataSize = result.size();
        Assert.assertEquals(32, dataSize);
    } catch (Exception e) {
        Assert.assertTrue(false);
    }
}

@MediumTest
public void testGetConstellationsNameAndPosition() {
    try {
        HashMap<String, double[]> result =
            cst_dao.getConstellationsNameAndPosition();
        int dataSize = result.size();
        Assert.assertEquals(32, dataSize);
    } catch (Exception e) {
        Assert.assertTrue(false);
    }
}
```

#### 4.2.1.2 Stars\_daoTest

Dans les tests ci-dessous, la valeur attendue doit correspondre au nombre d'étoiles présentes au sein de la table "stars".

```
@MediumTest
public void testGetStarsPosition() {
    try {
        ArrayList<double[]> result = stars_dao.getStarsPosition(6.0);
        int dataSize = result.size();
        Assert.assertEquals(1628, dataSize);
    } catch (Exception e) {
        Assert.assertTrue(false);
    }
}
```

Le tests ci-dessous vérifie si il n'y a pas d'exception déclenchée.

```
@MediumTest
public void testGreekLetterPosition() {
    try {
        for (CstName cstName: CstName.values()) {
            HashMap<String, double[]> result =
                stars_dao.greekLetterPosition(cstName.name());
        }
        Assert.assertTrue(true);
    } catch (Exception e) {
        Assert.assertTrue(false);
    }
}
```

Dans le tests ci-dessous, la valeur attendue doit correspondre au nombre d'étoiles nommées présentes au sein des constellations.

```
@MediumTest
public void testGetStarsNameAndPosition() {
    try {
        HashMap<String, double[]> result =
            stars_dao.getStarsNameAndPosition();
        int dataSize = result.size();
        Assert.assertEquals(48, dataSize);
    } catch (Exception e) {
        Assert.assertTrue(false);
    }
}
```

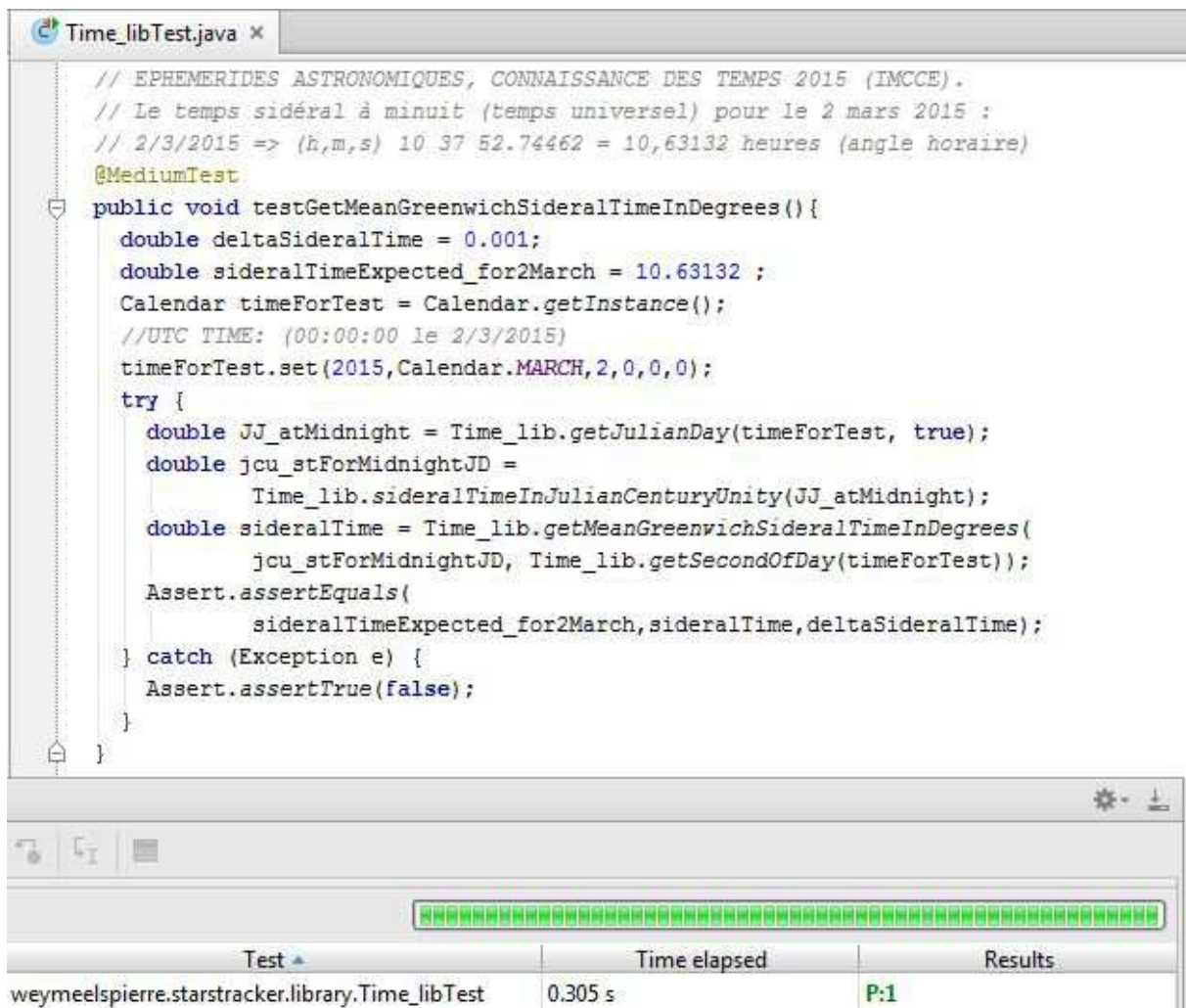
Pour conclure, le test concernant la classe "Ephemeris\_dao" fait partie de manière implicite au test de classe du modèle "Ephemeris".



## 4.2.2 Test unitaire de la classe Time\_lib

Le test ci-dessous démontre la précision du calcul du temps sidéral.

Pour le 2 mars 2015 à 1 : 00 (00 : 00 UTC) :



The screenshot shows an IDE window titled "Time\_libTest.java". The code is a Java test method for the `Time_lib` class. It uses `@MediumTest` and `Assert` to verify the calculation of sidereal time. The test sets a date of March 2, 2015, at 00:00 UTC and compares the calculated sidereal time against an expected value with a tolerance of 0.001 degrees.

```
// EPHEMERIDES ASTRONOMIQUES; CONNAISSANCE DES TEMPS 2015 (IMCCE).  
// Le temps sidéral à minuit (temps universel) pour le 2 mars 2015 :  
// 2/3/2015 => (h,m,s) 10 37 52.74462 = 10,63132 heures (angle horaire)  
@MediumTest  
public void testGetMeanGreenwichSideralTimeInDegrees(){  
    double deltaSideralTime = 0.001;  
    double sideralTimeExpected_for2March = 10.63132 ;  
    Calendar timeForTest = Calendar.getInstance();  
    //UTC TIME: (00:00:00 le 2/3/2015)  
    timeForTest.set(2015,Calendar.MARCH,2,0,0,0);  
    try {  
        double JJ_atMidnight = Time_lib.getJulianDay(timeForTest, true);  
        double jcu_stForMidnightJD =  
            Time_lib.sideralTimeInJulianCenturyUnity(JJ_atMidnight);  
        double sideralTime = Time_lib.getMeanGreenwichSideralTimeInDegrees(  
            jcu_stForMidnightJD, Time_lib.getSecondOfDay(timeForTest));  
        Assert.assertEquals(  
            sideralTimeExpected_for2March,sideralTime,deltaSideralTime);  
    } catch (Exception e) {  
        Assert.assertTrue(false);  
    }  
}
```

Below the code editor, a progress bar shows the test execution status. At the bottom, a table summarizes the test results:

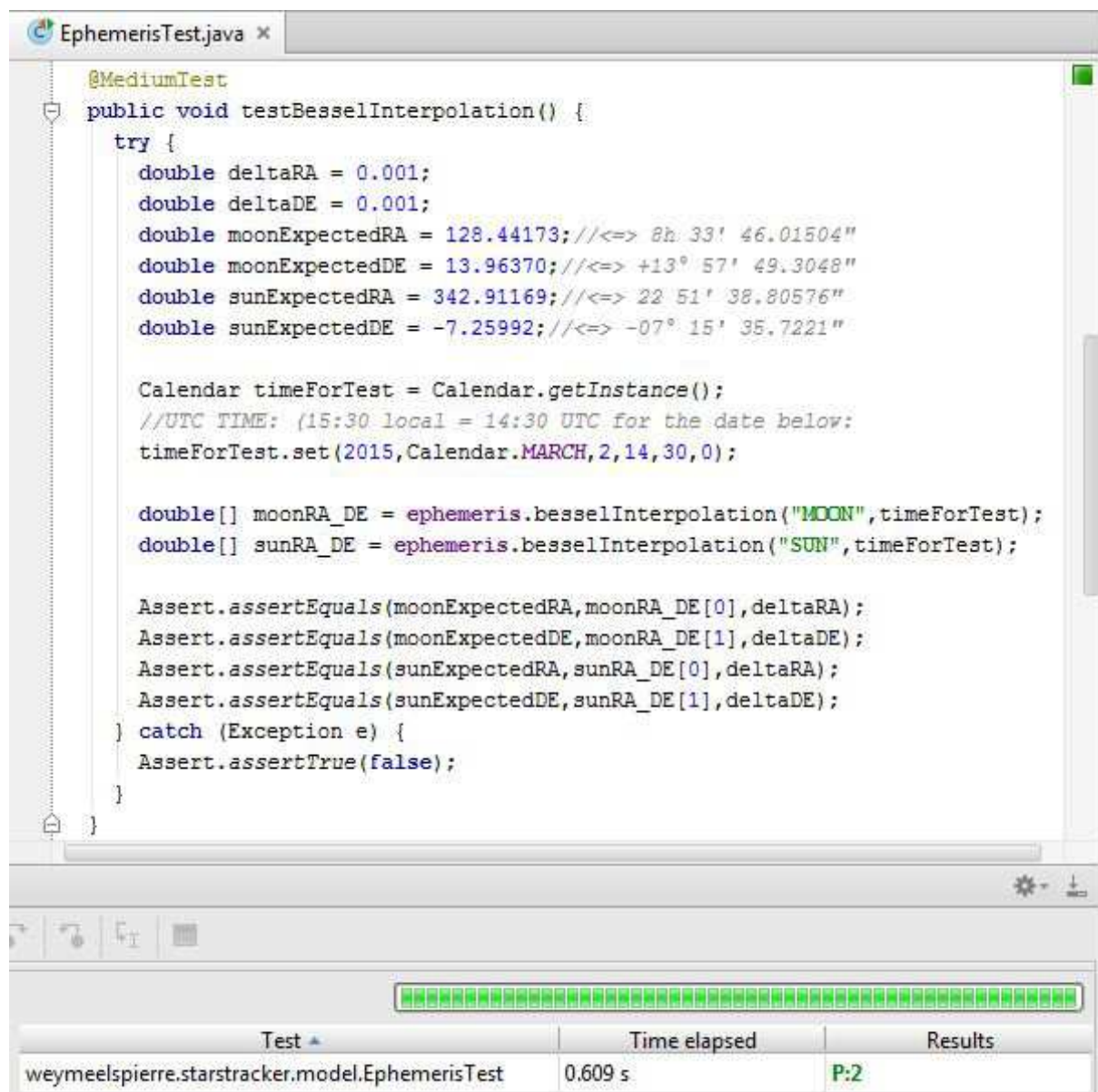
Test	Time elapsed	Results
weymeelspierre.startracker.library.Time_libTest	0.305 s	P:1

FIGURE 4.1 – Test de précision du calcul du temps sidéral.

### 4.2.3 Test unitaire de la classe Ephemeris

Le test ci-dessous démontre la précision du calcul de la position pour la Lune et le Soleil.

Pour le 2 mars 2015 à 15 : 30 (14 : 30 UTC) :



The screenshot shows an IDE window titled 'EphemerisTest.java'. The code defines a test method `testBesselInterpolation()` that uses `Calendar` to set a specific date and time, then calls `ephemeris.besselInterpolation()` for 'MOON' and 'SUN'. It asserts that the results match expected values within a delta of 0.001. Below the code editor, a progress bar is shown, and a table displays the test results.

```
@MediumTest
public void testBesselInterpolation() {
    try {
        double deltaRA = 0.001;
        double deltaDE = 0.001;
        double moonExpectedRA = 128.44173; //<=> 8h 33' 46.01504"
        double moonExpectedDE = 13.96370; //<=> +13° 57' 49.3048"
        double sunExpectedRA = 342.91169; //<=> 22 51' 38.80576"
        double sunExpectedDE = -7.25992; //<=> -07° 15' 35.7221"

        Calendar timeForTest = Calendar.getInstance();
        //UTC TIME: (15:30 local = 14:30 UTC for the date below:
        timeForTest.set(2015, Calendar.MARCH, 2, 14, 30, 0);

        double[] moonRA_DE = ephemeris.besselInterpolation("MOON", timeForTest);
        double[] sunRA_DE = ephemeris.besselInterpolation("SUN", timeForTest);

        Assert.assertEquals(moonExpectedRA, moonRA_DE[0], deltaRA);
        Assert.assertEquals(moonExpectedDE, moonRA_DE[1], deltaDE);
        Assert.assertEquals(sunExpectedRA, sunRA_DE[0], deltaRA);
        Assert.assertEquals(sunExpectedDE, sunRA_DE[1], deltaDE);
    } catch (Exception e) {
        Assert.assertTrue(false);
    }
}
```

Test	Time elapsed	Results
weymeelspierre.startracker.model.EphemerisTest	0.609 s	P:2

FIGURE 4.2 – Test de précision de calcul des éphémérides.

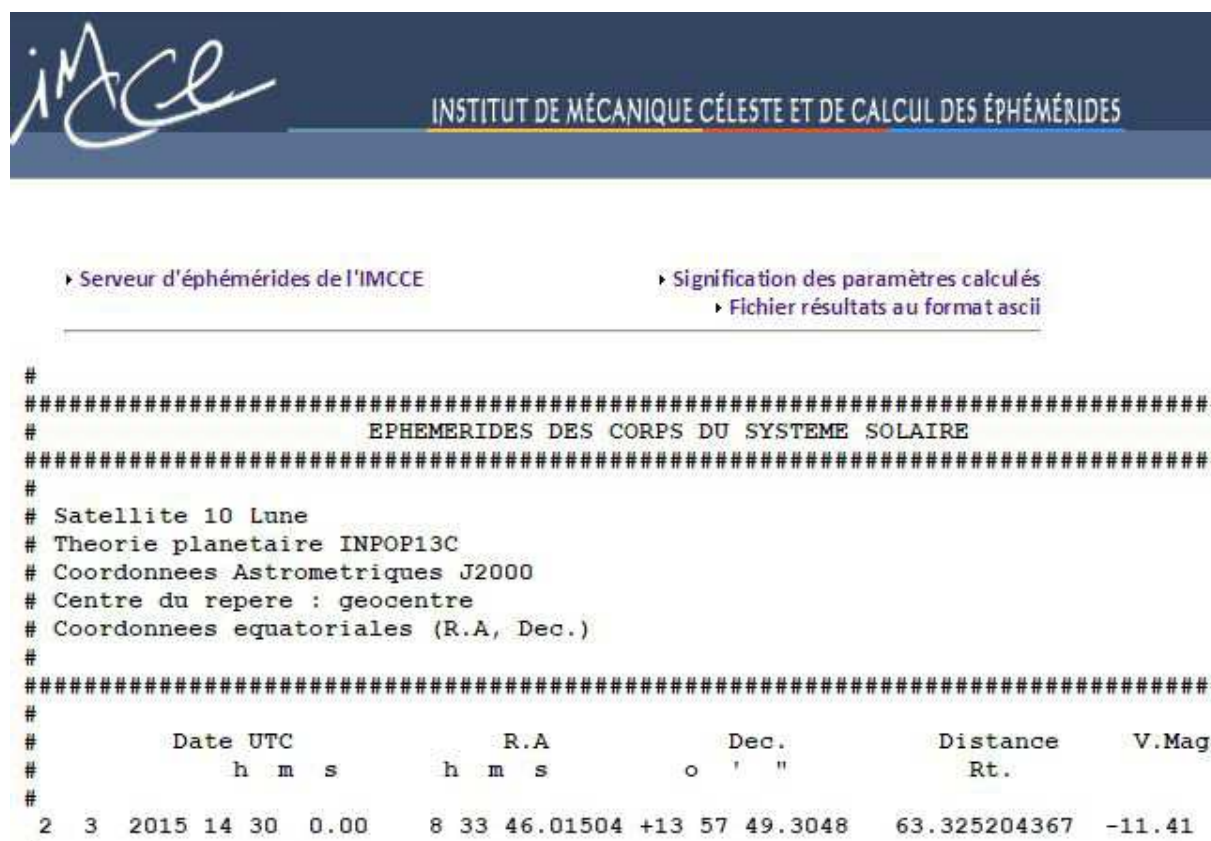


FIGURE 4.3 – Origine de la position lunaire utilisée pour le test.

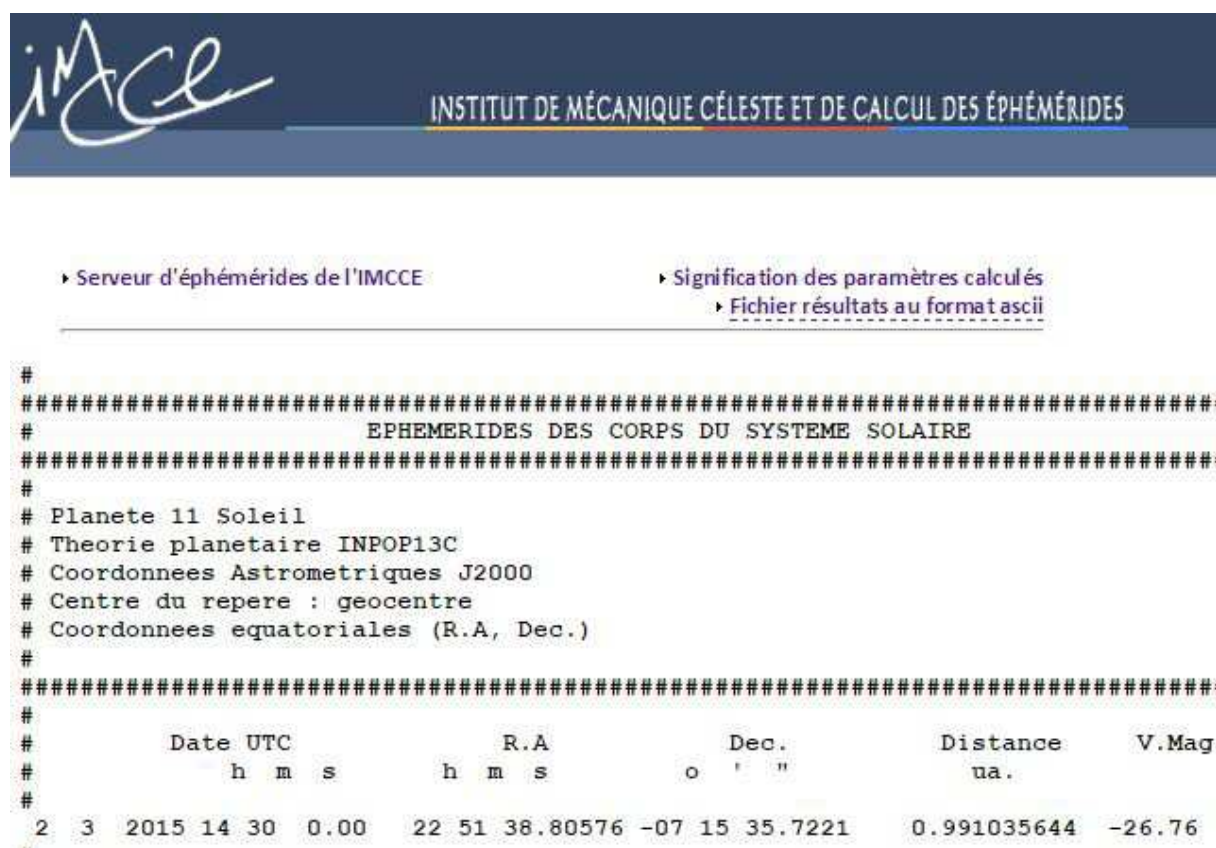


FIGURE 4.4 – Origine de la position solaire utilisée pour le test.

**Troisième partie**

**Conclusion**

# Conclusion

Dans le cadre de ce travail de fin d'étude, j'ai été amené à étendre mon domaine de connaissance à l'utilisation de l'interface de programmation OpenGL et Android.

Ces deux dernières couvrant chacun un domaine particulièrement vaste en terme de nouvelles notions, cette expérience fut intellectuellement très enrichissante.

De plus, le projet étant multidisciplinaire, les difficultés rencontrées ont été nombreuses et ont sollicitées la majeure partie des compétences en analyse et en programmation acquises lors de ma formation.

Ainsi, sans ces nouvelles compétences, et sans ma passion dévorante pour l'astronomie, ce projet n'aurait jamais pu voir le jour.

Pour conclure, cette application peut continuer à évoluer avec des fonctionnalités supplémentaires qui sont développées au chapitre concernant les améliorations futures <sup>1</sup>.

---

1. voir en annexe C.

## Quatrième partie

### Annexe

# Annexe A

## Rendu final

Hormis l'écran d'initialisation, les images ci-dessous ont été prises en fausses couleurs et ont été doublé en taille (en pixel) des éléments afin de pouvoir distinguer les détails (effacés par le fond noir d'origine et l'absence de rétroéclairage).

Les vraies couleurs sont :

- le blanc au lieu du noir pour les étoiles ;
- le gris au lieu du bleu pour la grille ;
- le noir au lieu du gris clair pour le fond ;
- le gris au lieu du kaki pour la lune.



FIGURE A.1 – Ecran d'initialisation.

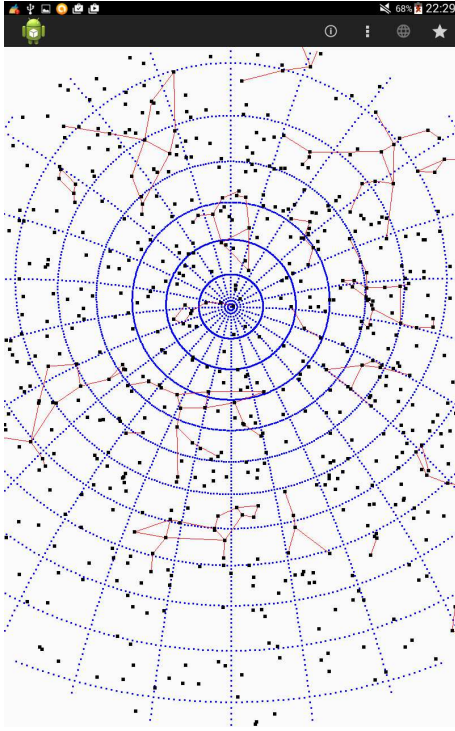


FIGURE A.2 – Vue du pôle nord.

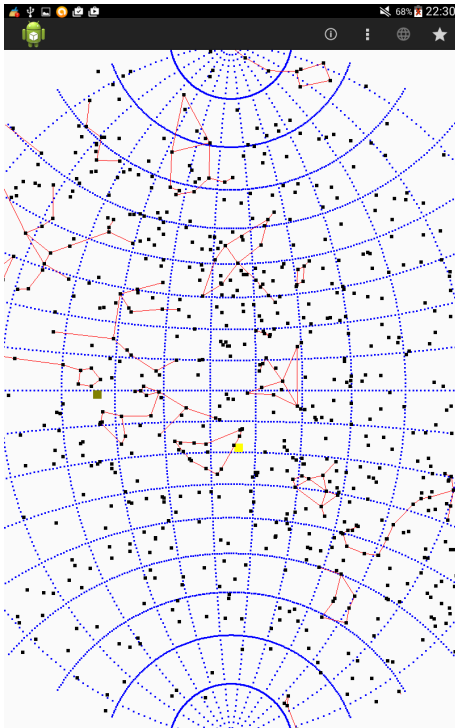


FIGURE A.3 – Vue équatoriale avec la Lune et le Soleil.



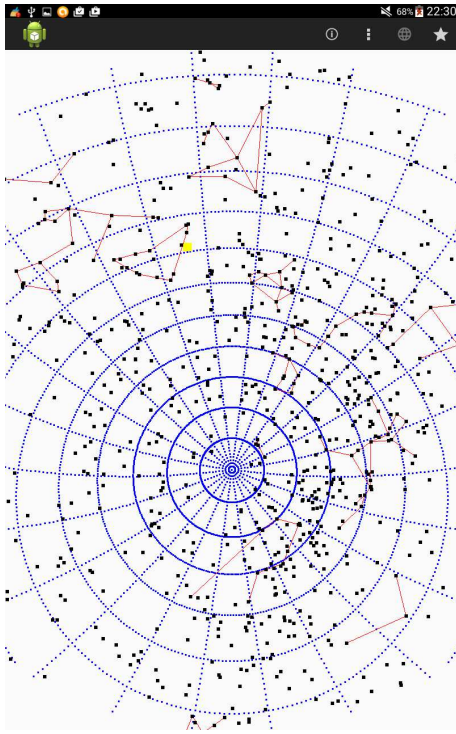


FIGURE A.4 – Vue du pôle sud avec le Soleil.

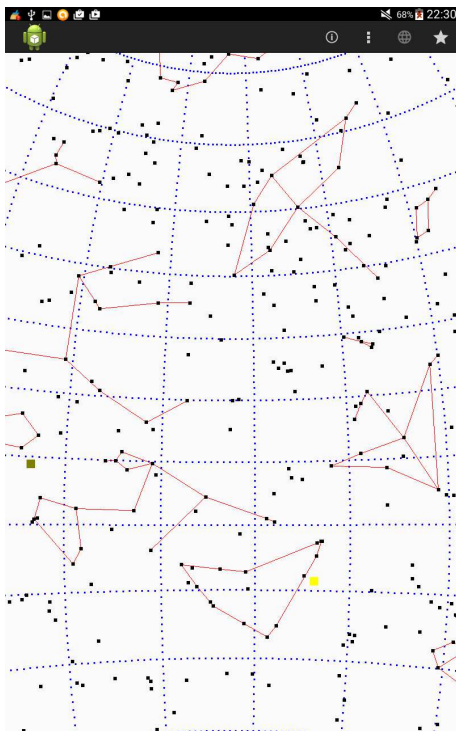


FIGURE A.5 – Vue de la Lune et du Soleil.

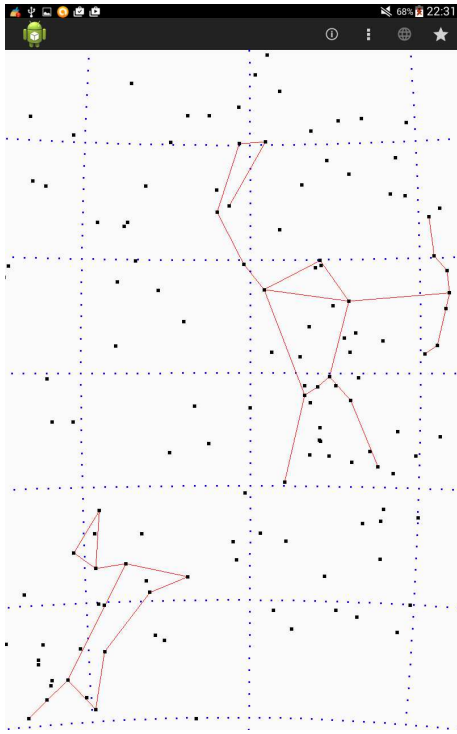


FIGURE A.6 – Vue "zoomée" des constellations Orion et Canis mayor.

## Annexe B

# Représentation des classes logicielles et des packages

### B.1 Remarques concernant le diagramme de classes logicielles :

La classe "OrientationHac", ne concerne pas l'utilisation actuelle de l'application.

Néanmoins, elle existe car elle concerne des développements futurs qui n'ont pas pu être menés à leur terme dans le cadre de cette épreuve<sup>1</sup>.

---

1. cf. annexe C



## B.2 Diagramme de classes logicielles

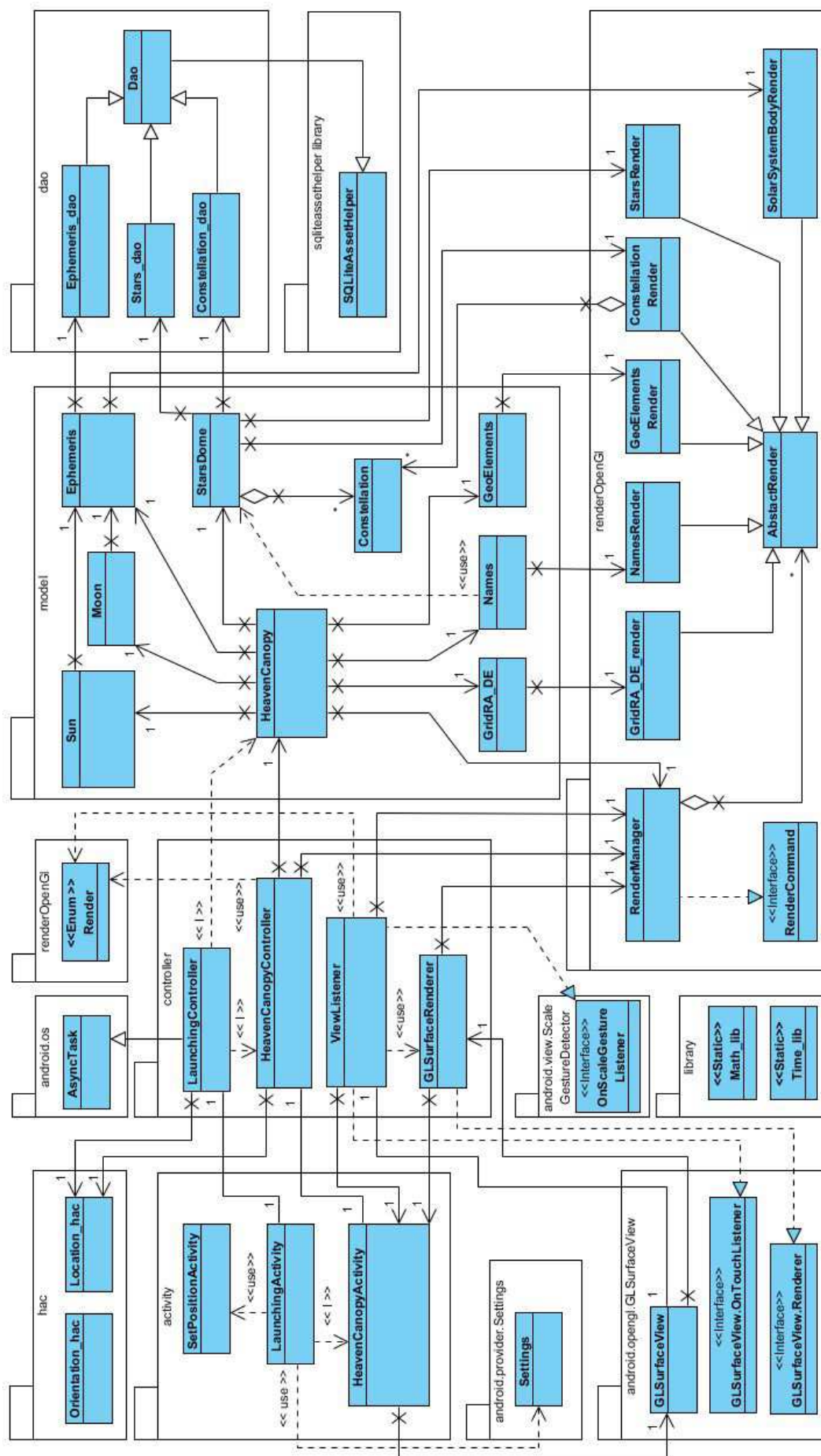


FIGURE B.1 – Diagramme de classes logicielles.

Légende => I : signifie "instancie". | use : signifie "utilise".

## B.3 Dictionnaire de classes

### B.3.1 Package activity



FIGURE B.2 – Package activity.

### B.3.2 Package controller



FIGURE B.3 – Package controller.

### B.3.3 Package model

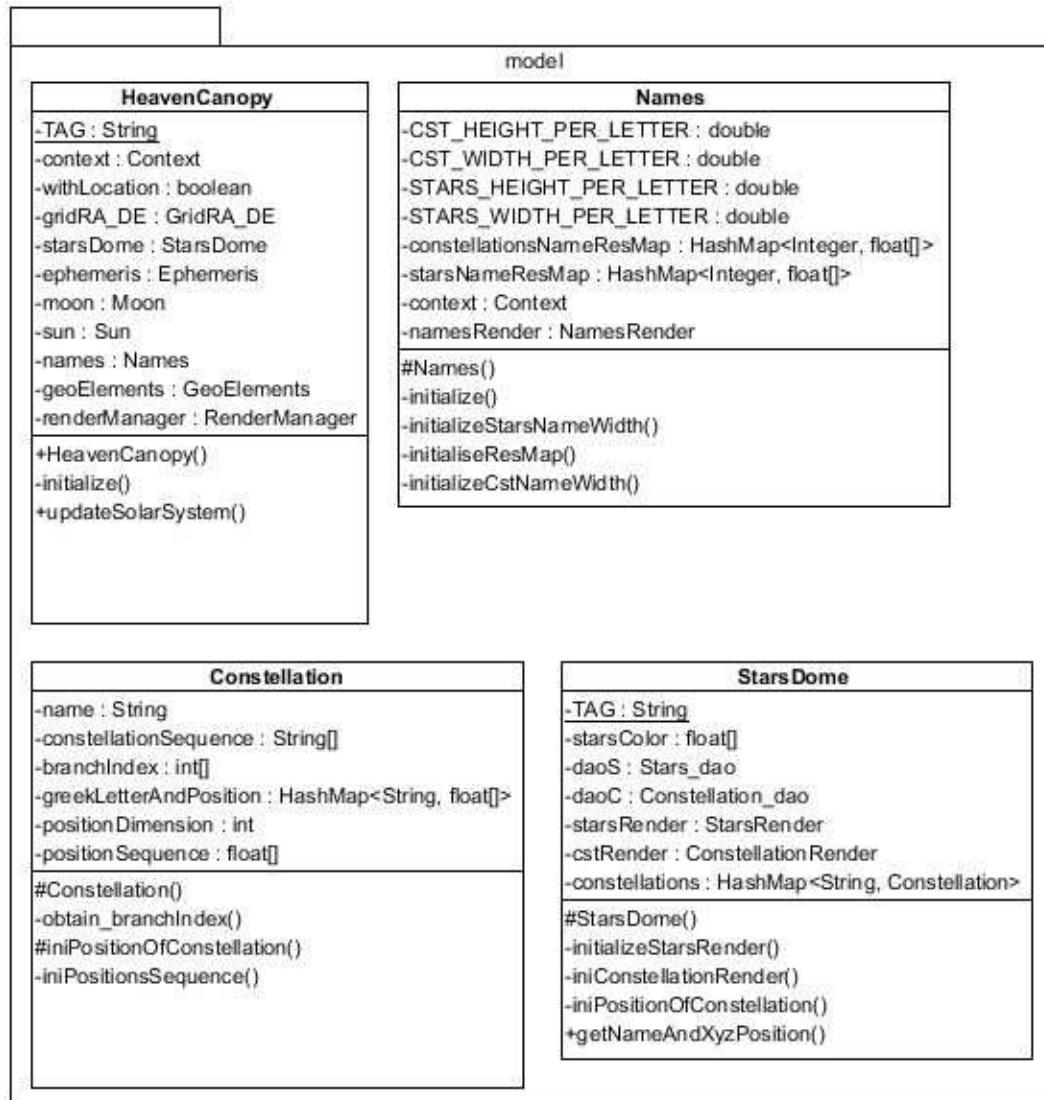


FIGURE B.4 – Package model, première partie.



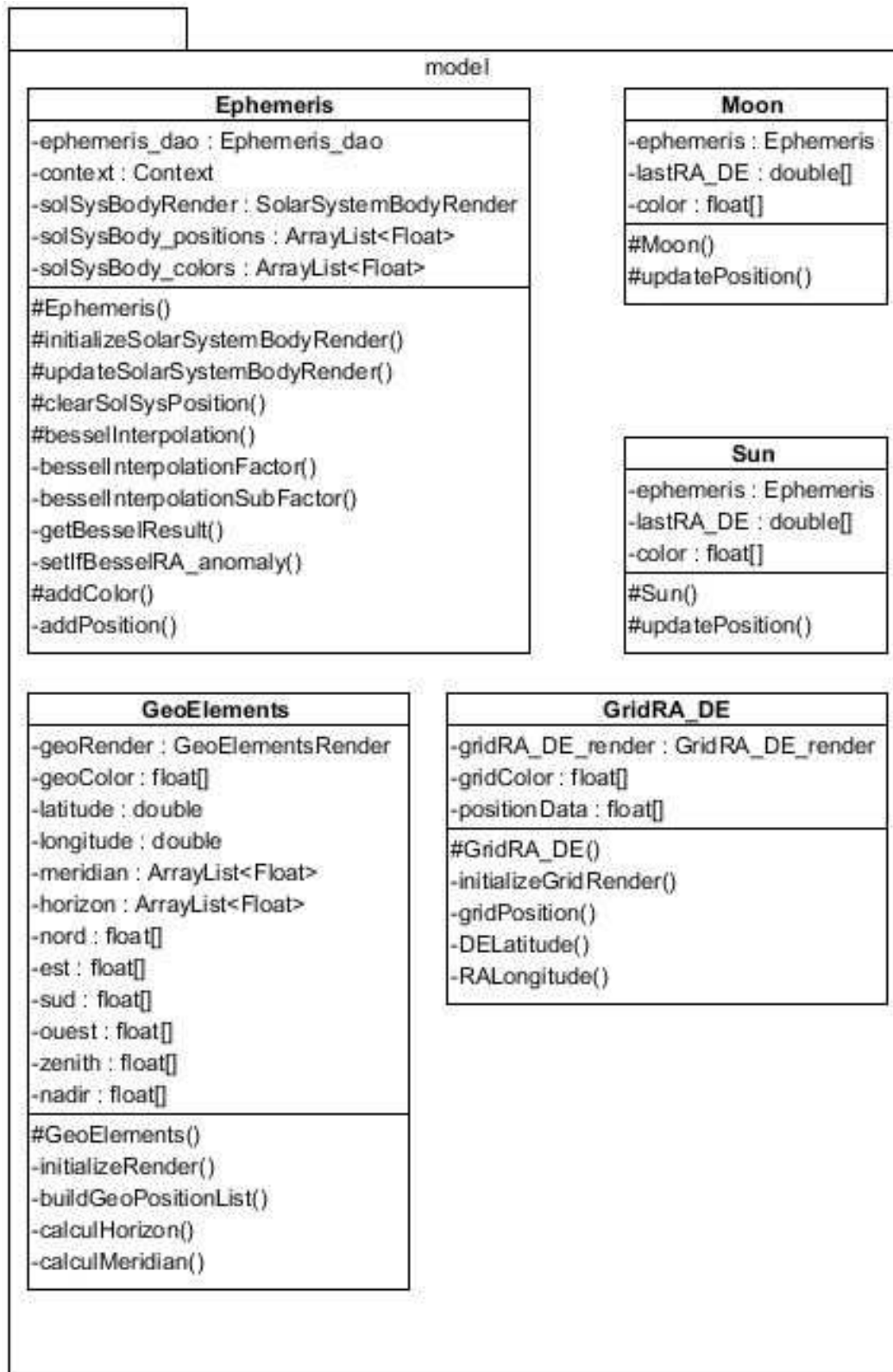


FIGURE B.5 – Package model, deuxième partie.

### B.3.4 Package dao

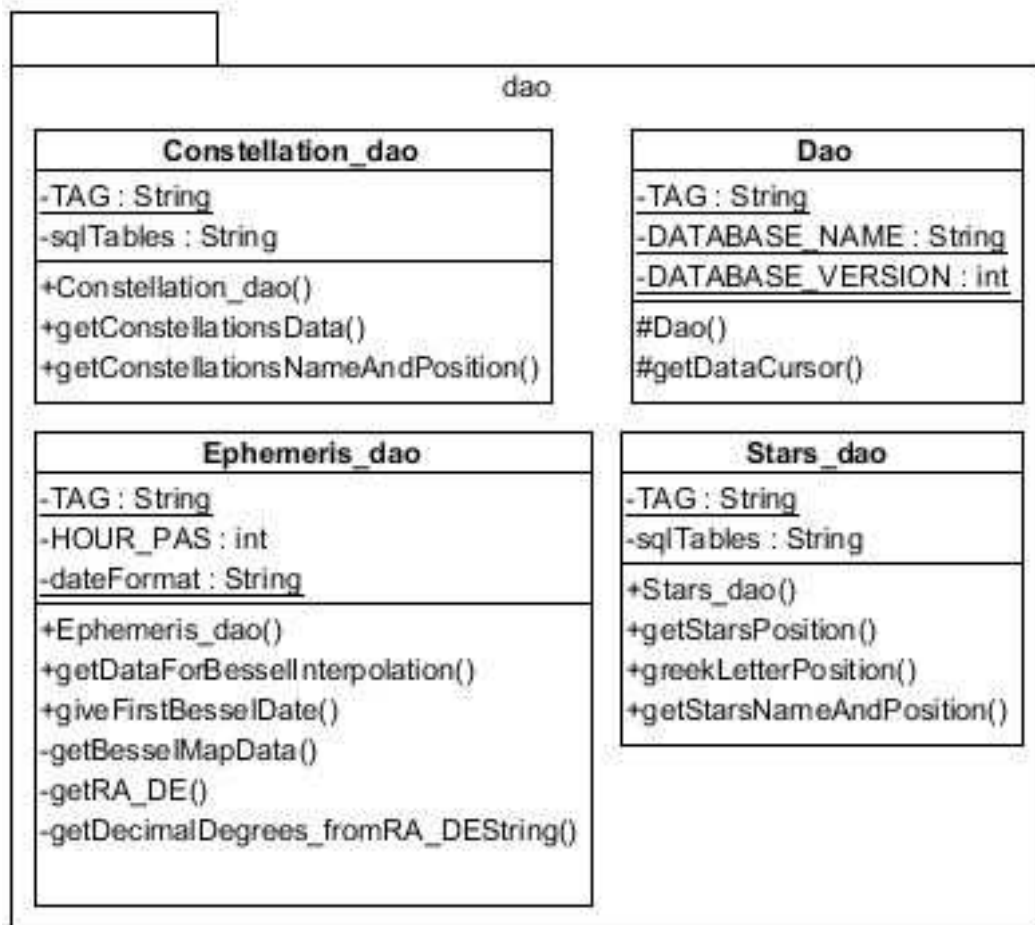


FIGURE B.6 – Package dao.

### B.3.5 Package renderOpenGL



FIGURE B.7 – Package renderOpenGL, première partie.

renderOpenGL			
	<b>NamesRender</b>	<b>GeoElementsRender</b>	<b>ConstellationRender</b>
	-TAG : String -context : Context -cst_texturesResAndBuffer : HashMap<Integer, FloatBuffer> -stars_texturesResAndBuffer : HashMap<Integer, FloatBuffer> -cst_texturesBitmapAndBuffer : HashMap<Bitmap, FloatBuffer> -stars_texturesBitmapAndBuffer : HashMap<Bitmap, FloatBuffer> -drawListBuffer : ShortBuffer -uvBuffer : FloatBuffer -indices : short[] -uvs : float[] -vertexShader : String -fragmentShader : String +NamesRender() -iniTexturesBufferNb() -getFloatPositionsBufferOff() -changeIntoShortBuffer() #setActive() -getTexturesBufferAndBitmap() -getBitmap() #iniProgramAndShader() #draw() -prepareTextureFrameForShader() -prepareTextureHandleForShader() -drawTextures() -preparePositionForShaderTxt() -setupImage()	-specificModelMatrix : float[] -TAG : String -geoColor : float[] -vertexShader : String -fragmentShader : String +GeoElementsRender() -prepareUniformColorsForShader() #setActive() #iniProgramAndShader() -getSpecificModelViewProjMatrix() #prepareSpecificMatrixMVPForShader() #draw()	-TAG : String -constellations : HashMap<String, Constellation> -bufferPositionsMap : HashMap<String, FloatBuffer> -vertexShader : String -fragmentShader : String +ConstellationRender() -getFloatPositionsBufferOff() -preparePositionForShader() #setActive() #iniProgramAndShader() #draw()
	<b>GridRA_DE_render</b>	<b>StarsRender</b>	<b>SolarSystemBodyRender</b>
	-TAG : String -gridColor : float[] -vertexShader : String -fragmentShader : String +GridRA_DE_render() -prepareUniformColorsForShader() #setActive() #iniProgramAndShader() #draw()	-TAG : String -starsColors : float[] -vertexShader : String -fragmentShader : String +StarsRender() -prepareUniformColorsForShader() #setActive() #iniProgramAndShader() #draw()	-TAG : String -vertexShader : String -fragmentShader : String +SolarSystemBodyRender() #setActive() #iniProgramAndShader() #draw() +update()

FIGURE B.8 – Package renderOpenGL, deuxième partie.

### B.3.6 Package library



FIGURE B.9 – Package library.

### B.3.7 Package hac

La classe "OrientationHac", ne concerne pas l'utilisation actuelle de l'application <sup>2</sup>.

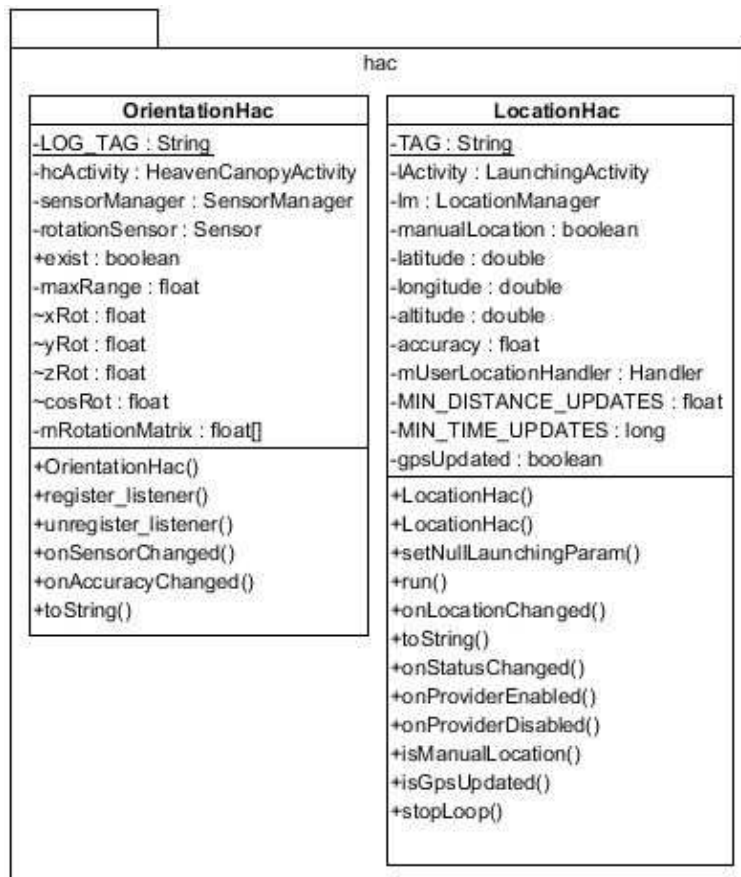


FIGURE B.10 – Package hac.

2. voir annexe C.

# Annexe C

## Améliorations futures

Le sujet abordé dans le cadre de ce projet, permet un nombre élevé d'idées pour de futures améliorations et fonctionnalités.

Par conséquent, la liste des intentions qui sont décrites ci-dessous, n'est pas exhaustive.

### C.1 Résolution des problèmes rencontrés

Les tests utilisateurs ont démontré les besoins suivant :

1. De pousser l'analyse en ce qui concerne la liaison avec le GPS car celui-ci ne provoque pas à chaque fois l'événement de mise à jour de la position.
2. D'analyser avec plus d'attention la gestion des ressources mémoires utilisé par l'environnement OpenGL, et de réduire l'impact du problème rencontré lors de l'affichage des noms.

### C.2 Amélioration du rendu visuel

Les échanges entre le programme OpenGL (CPU) et les shaders (GPU), sont par définition assez lents, il serait donc utile de mieux exploiter le pipeline de rendu ainsi que les shaders associés à ce dernier. Nous pouvons aussi envisager les évolutions graphiques suivantes :

**Aspect des étoiles :**

un point lumineux dont la luminosité serait adaptée à sa magnitude et sa couleur à son type spectral pourrait être attribué à chaque étoile pour les distinguer.

**Aspect des corps célestes :**

faire apparaître l'image du corps.

**Aspect des constellation :**

implémenter les 88 constellations et afficher les interconnexions entre elles.

### C.3 Couplage avec les données d'orientations

La classe OrientationHac à été implémentée dans ce but, elle permet d'obtenir les données suivantes :

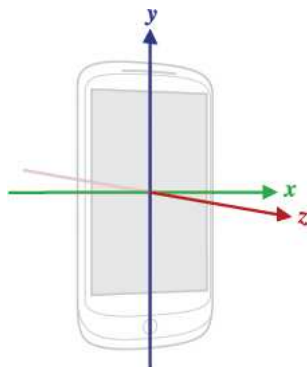


FIGURE C.1 – Repère de coordonnées du mobile.

<http://developer.android.com/reference/android/hardware/SensorEvent.html#values>

**Azimuth :**

angle entre le nord magnétique et l'axe y, par rotation autour de l'axe z.

**Pitch (tangage) :**

angle de rotation autour de l'axe x.

**Roll (roulis) :**

angle de rotation autour de l'axe y.

Ces données proviennent de l'écoute des matériels hardwares suivants : gyroscope, de l'accéléromètre et du capteur géomagnétique.

Pour conclure, ce qui a été décrit ci-dessus sera utile pour l'élaboration d'un mode de visualisation tenant compte du positionnement physique de l'utilisateur.

## C.4 Mode couplage orientation-position

Grâce à la récolte de données fournies par le GPS, le gyroscope, l'accéléromètre et le capteur géomagnétique.

Ce mode de visualisation, afficherait automatiquement la partie de la voûte correspondant à l'orientation physique de l'écran et à la position géographique de l'utilisateur.



# Annexe D

## Android

(la description présente dans ce chapitre est un résumé de la théorie présente sur le site "Android Developers"<sup>1</sup>).

### D.1 Description

Android est un système d'exploitation open source pour appareil mobile. Il est actuellement le plus utilisé au monde et il existe sous plusieurs versions, chacune apportant de nouvelles fonctionnalités.

Par souci de clarté, la description ci-dessous s'attarde sur les éléments théoriques qui sont des caractéristiques importantes d'un programme android.

### D.2 Android et le langage de Java

Selon Wikipédia, la bibliothèque Java d' Android est similaire à J2SE (Java Standard Edition).

La différence majeure se situe au niveau de l'interface graphique, où Android utilise ses propres librairies au lieu de Swing et AWT.

Autre différence, sous Android chaque programme est exécuté par une machine virtuelle appelée Dalvik. Avant que le code Java puisse être interprété par cette dernière, le bytecode, préalablement créé par un compilateur pour machine virtuelle Java, doit être recompilé par un compilateur approprié qui le transformera sous forme d'un fichier (.dex) exécutable par Dalvik.

### D.3 La classe abstraite Context

Cette classe permet d'accéder aux informations concernant l'environnement de l'application.

---

1. cf. bibliographie [2].

## D.4 La classe Activity

Cette classe assure la gestion de l’affichage et des événements de l’interface utilisateur.

Cette interface utilisateur est décrite au sein d’un fichier xml dont la liaison avec la classe est assurée par la fonction "setContentView(View)".

Ce qui caractérise cette classe, c’est son cycle de vie :

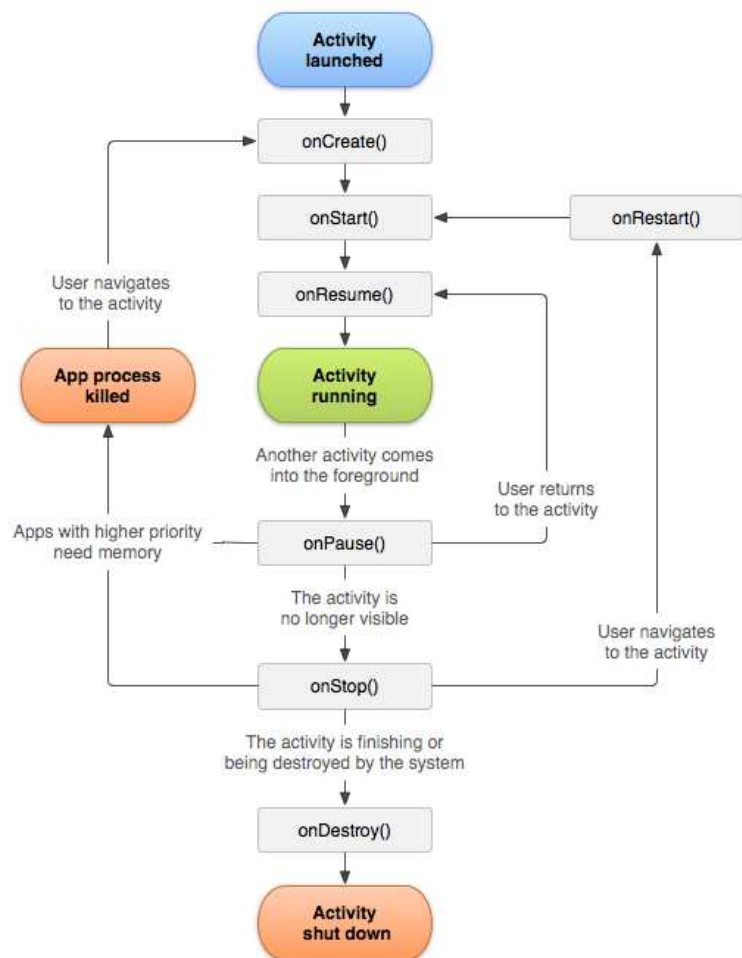


FIGURE D.1 – Cycle de vie

<http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle>

### D.4.1 Description des méthodes

**onCreate()** : appelée à la création de l’activité. Utile pour initialiser l’interface graphique.

**onStart()** : appelée au moment où l’interface graphique devient visible.

**onResume()** : appelée au moment où l’interaction avec l’interface graphique devient possible.

**onPause()** : appelée avant que le système bascule sur une autre activité. La nouvelle activité couvre partiellement la précédente.

**onStop()** : appelée au moment où l’activité n’est plus visible. La nouvelle activité couvre totalement la précédente.

**onRestart()** : appelée si l’activité revient et propose à nouveau une interaction avec son interface graphique.

**onDestroy()** : appelée juste avant la destruction de l’activité.

## D.5 La classe "R"

La classe "R" est une classe générée à la compilation et qui possède comme constantes statiques les identifiants des ressources déclarées sous le format xml.

Ces constantes permettent aux différentes classes java d’accéder aux différentes ressources via l’identifiant statique de celles-ci.

## D.6 Le dossier "res"

Ce dossier réunit toutes les ressources nécessaires à l’affichage graphique.

Par conséquent, les ressources sont réparties dans plusieurs sous-dossier en fonction du type, de la langue utilisée par l’application, de la densité ou de la taille de l’écran.

Ainsi, cela permet au système de choisir automatiquement la ressource adaptée au mobile sur lequel tourne l’application.

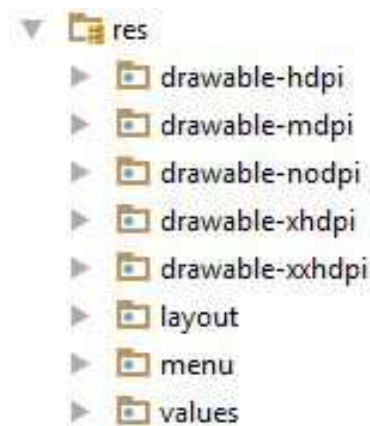


FIGURE D.2 – Le dossier ressource

### D.6.1 Description des sous-dossiers

**drawable** : contient les éléments de type image (bitmap,png,jpeg,...).

**layout** : contient les fichiers xml qui décrivent la structure et les éléments des différentes interfaces graphiques utilisées par le programme.

**menu** : contient les structures des interfaces graphiques destinées aux menus (par exemple, la barre verticale située en haut de l’écran).

**values** : contient les fichiers xml qui soit définit les paramètres de style (couleur,...), soit les valeurs de certaines constantes de type String

## D.7 Le fichier "manifest"

Le manifest est un fichier xml qui décrit les besoins en terme de composants et les limites spécifiques pour que l’application puisse fonctionner.

Par conséquent, c’est par ce fichier que les autorisations spécifiques seront accordées par le système Android à ladite application.

De plus, ce fichier permet à Google Play de montrer l’application seulement aux utilisateurs possédant la bonne configuration matérielle.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="weymeelspierre.starstracker" >

    <uses-sdk
        android:minSdkVersion="15"
        android:targetSdkVersion="16" />

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <uses-feature
        android:name="android.hardware.location.gps"
        android:required="true" />
    <uses-feature
        android:glEsVersion="0x00020000"
```

```

        android:required="true" />
<uses-feature
    android:name="android.hardware.sensor.accelerometer"
    android:required="false" />
<uses-feature
    android:name="android.hardware.sensor.compass"
    android:required="false" />
<uses-feature
    android:name="android.hardware.sensor.gyroscope"
    android:required="false" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".activity.LaunchingActivity"
        android:label="@string/app_name"
        android:theme="@style/AppTheme"
        android:screenOrientation="portrait">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name=".activity.HeavenCanopyActivity"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
    </activity>
    <activity
        android:name=".activity.SetPositionActivity"
        android:label="@string/app_name"
        android:screenOrientation="portrait"
        android:noHistory="true">
    </activity>
</application>

</manifest>

```

### D.7.1 Description du manifest

**xmlns** : Cet attribut définit l'espace de nom Android.

**uses-sdk** : Exprime la version minimale et celle que cible l'application.

**uses-permission** : Cet attribut définit les permissions dont l'application a besoin.

**uses-feature** : Cet attribut définit les composants de type hardware et software dont l'application a besoin.

**application** : C'est ici que l'on déclare les composants de l'application (icône, activité, service,...).

# Annexe E

## OpenGL

(La description présente dans ce chapitre est un résumé de la théorie présente sur le site [openGl.org](http://openGl.org)<sup>1</sup> et de celle présente sur le site de Song Ho Ahn<sup>2</sup>.)

### E.1 Description

OpenGL est une librairie de fonctions open source pour le calcul et le rendu d'images en deux ou en trois dimensions . Pour être le plus concis possible, l'impasse a été faite sur les explications des étapes non obligatoires du pipeline de rendu.

### E.2 Les différentes versions

OpenGL existe sous plusieurs versions de 1.0 à 4.5 . Les versions adaptées pour les appareils mobiles sont appelées OpenGL ES.

**OpenGL ES 1.0 et 1.1** : sont supportées à partir de la version 1.0 d'Android.

Mais, ces méthodes sont de plus en plus dépréciées.

**OpenGL ES 2.0** : dérivée d'OpenGL 2.0, elle est supportée à partir de la version 2.2 d'Android.

Cette version 2.0 apporte la possibilité de coder en "OpenGL Shading Language" (GLSL), un langage qui permet la programmation d'éléments appelés "shader"<sup>3</sup>.

Nous pouvons donc intervenir plus intimement dans le "Rendering Pipeline" de la carte graphique.

**OpenGL ES 3.0** : dérivée d'OpenGL 3.3 et 4.2, elle est supportée à partir de la version 4.3 d'Android.

**OpenGL ES 3.1** : est supportée à partir de la version 5.0 d'Android.

### E.3 Les primitives

Les primitives sont des formes géométriques de base et sont interprétés comme une suite spécifique de sommets (vertex) par le pipeline de rendu.

Les principales primitives sont :

**GL\_POINTS** : un point,

chaque sommet du buffer d'entrée sera considéré individuellement par le pipeline de rendu.

**GL\_LINES** : une ligne, ensemble de deux sommets.

**GL\_TRIANGLES** : un triangle, ensemble de trois sommets.

**GL\_QUADS** : un carré, ensemble de quatre sommets.

---

1. cf. bibliographie [15].    2. cf. bibliographie [1].    3. cf. définition en annexe G.

## E.4 Rendering Pipeline

Pour permettre le rendu d'objets graphiques (bitmaps, points, polygones, vecteurs,...), OpenGL suit une séquence linéaire d'étapes appelée le "Rendering Pipeline".

Concernant les données traitées, elles circulent entre ces étapes sous la forme d'un buffer<sup>4</sup>.

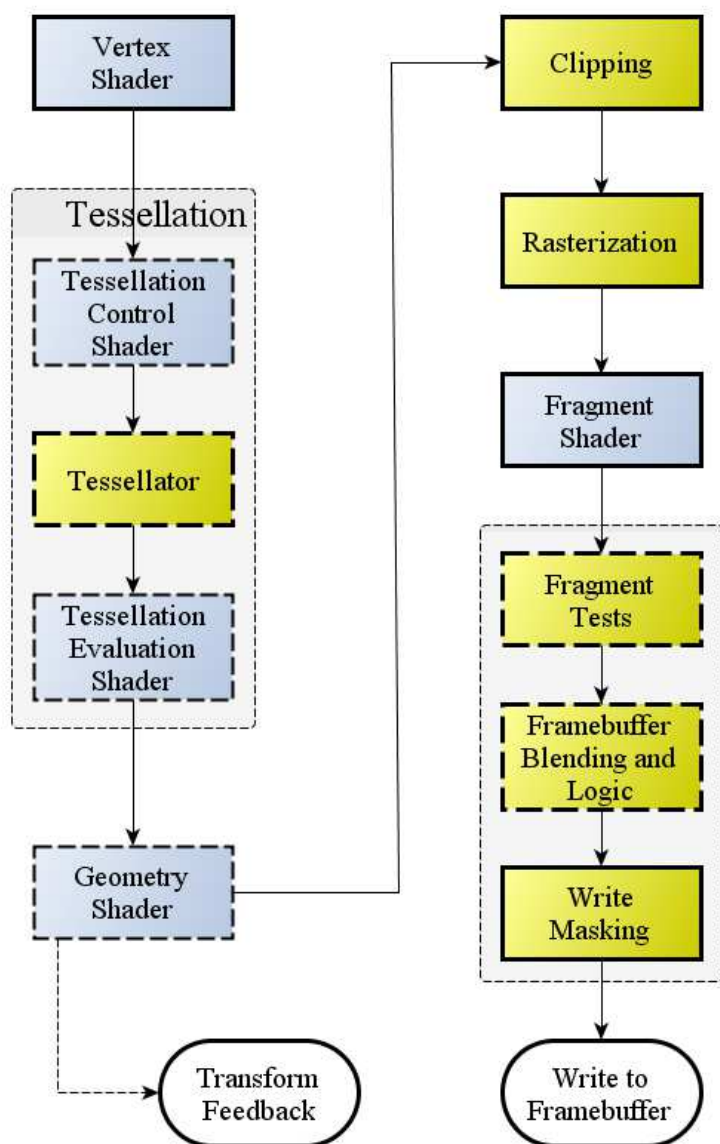


FIGURE E.1 – le pipeline de rendu OpenGL

[https://www.opengl.org/wiki/Rendering\\_Pipeline\\_Overview](https://www.opengl.org/wiki/Rendering_Pipeline_Overview)

### E.4.1 Vertex Shader

Processus<sup>5</sup> responsable du traitement individuel de chaque vertex<sup>6</sup>.

Il effectue en général les calculs de transformation de coordonnée via les matrices décrites dans ce chapitre et des transformations de l'espace de post-projection.

Il peut aussi être utilisé pour le rendu de l'éclairage par vertex.

Il traite un flux de vertex en entrée possédant généralement les attributs suivants :

**coordonées de position** : généralement sous la forme d'un vecteur de dimension deux ou trois.

**gl\_VertexID** : l'indice du vertex en cours de traitement.

**gl\_InstanceID** : l'indice du vertex en cours de traitement dans le cas d'un rendu instancié.

Le flux de sortie est composé des attributs suivants (le rapport entre le flux d'entrée et de sortie est de 1 :1) :

4. cf. définition en annexe G. 5. lié à un programme appelé Vertex Shader, voir la définition de "shader" en annexe G.  
6. cf. définition en annexe G.

**vec4 gl\_Position** : les coordonnées "clip-space" de chaque vertex.

**float gl\_PointSize** ? : la taille, largeur et hauteur en pixel. Valable uniquement pour les primitives de type point.

**float gl\_ClipDistance** ? : la distance du vertex par rapport au "clip-plane". Une valeur positive indique qu'il se situe à l'arrière de ce plan.

## E.4.2 Tessellation

Dans cette partie du pipeline (optionnelle et disponible à partir de la version 4.0), on utilise un type de primitive secondaire appelé "patche" dont le nombre de sommets a été préalablement précisé.

La "Tessellation" consiste à diviser en primitives plus petites ces "patches" et à les traiter.

**Tessellation Control Shader (TCS)** : détermine le nombre de sous-primitives et effectue certaines opérations sur le "patche".

**Tessellator** : génère les sous-primitives.

**Tessellation Evaluation Shader (TES)** : opère sur les coordonnées abstraites des sous-primitives.

## E.4.3 Geometry Shader

Cette étape (optionnelle) permet de manipuler, effacer ou de convertir en un autre type chaque primitive individuellement.

## E.4.4 Transform Feedback

Les primitives sortantes du shader de géométrie ou d'assemblage (les primitives sont assemblées juste après le vertex shader), sont disponibles sous la forme d'une série de "buffer" spécifiques.

Cela permet une utilisation ultérieure de ces résultats.

## E.4.5 Clipping (and "Face culling")

Ce processus coupe les primitives entrantes pour les faire correspondre au volume de la vue. Pour cela, il utilise les coordonnées de type "clip-space" fournies par le vertex shader.

Le volume de visualisation pour un sommet est le suivant (cf. pipeline de transformation) :

$$\begin{aligned} -w_c &\leq x_c \leq w_c \\ -w_c &\leq y_c \leq w_c \\ -w_c &\leq z_c \leq w_c \end{aligned}$$

Pour conclure, c'est aussi au sein de ce processus que le "Face Culling" est effectué sur les primitives de type triangle.

Le face culling consiste à éliminer les dites primitives qui ne présentent pas leur face apparente.

## E.4.6 Rasterization

Ce processus trame<sup>7</sup> les primitives entrantes.

Le résultat du tramage d'une primitive est une suite de fragment<sup>8</sup>

## E.4.7 Fragment Shader

Processus<sup>9</sup> qui attribue pour chaque fragment (= pixel) entrant une couleur.

Chaque fragment entrant possède les attributs suivants (liste non exhaustive) :

**vec4 gl\_FragCoord** : les coordonnées du fragment dans le système de référence "window-space".

**bool gl\_FrontFacing** : égal à "true" si le fragment provient de la face avant ("front-face") de la primitive lui donnant naissance, "false" dans le cas de la face arrière ("back-face").

Toujours égal à "true" si cette primitive n'était pas un triangle.

**vec2 gl\_PointCoord** : position relative du fragment vis-à-vis du coin supérieur gauche d'une primitive de type point.

Cette étape ajoute au flux de sortie (au moins) l'attribut suivant (pour OpenGL ES 2.0) :

**gl\_FragColor** : la couleur associée à chaque fragment.

7. cf. définition en annexe G.    8. cf. définition en annexe G.    9. lié à un programme appelé Fragment Shader, voir la définition de "shader" en annexe G.

## E.4.8 Opération par échantillon

(Cette étape est optionnelle.)

Pour chaque fragment entrant :

**Fragment Tests** : effectue sur le fragment une suite de test spécifiée à l'origine du processus de rendu.

Si le test échoue, le fragment sera retiré du "framebuffer".

**Framebuffer Blending and Logic** : mélange, manipule la couleur associée au fragment.

**Write Masking** : Ce processus peut interdire certains composants d'un fragment d'être écrits dans le "framebuffer".

## E.5 Le pipeline de transformation

Les coordonnées de chaque point qui compose un objet graphique subissent un certain nombre de transformations avant que ledit point puisse être affiché à l'écran.

Ces différentes étapes suivent un schéma linéaire de ce type :

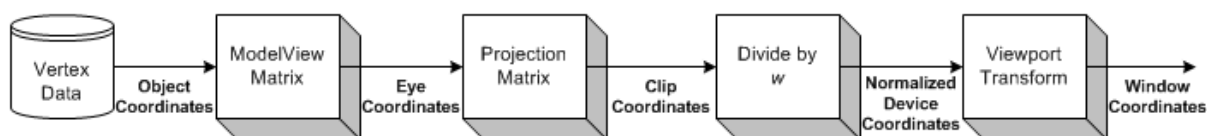


FIGURE E.2 – le pipeline de transformation OpenGL

[http://www.songho.ca/opengl/gl\\_transform.html](http://www.songho.ca/opengl/gl_transform.html)

### E.5.1 Object coordinates

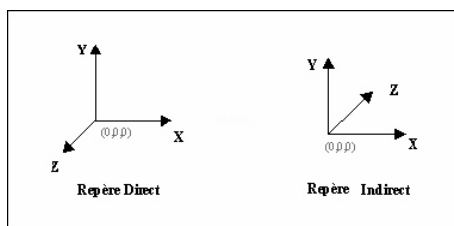


FIGURE E.3 – Repères direct et indirect

<http://ferretti.info/fabio/memoire/doc0002.htm>

Le système de référence d'axe orthonormés xyz du modèle graphique est un repère direct, tout comme celui de la vue.

Par souci de clarté, ce système a été choisi comme celui d'origine.

C'est à dire, celui de base  $(X, Y, Z) = ((1,0,0), (0,1,0), (0,0,1))$  et de centre  $O = (0,0,0)$ . Dans ce système, un vecteur de coordonnées objet est de dimension quatre :

$$\overrightarrow{V_{obj}} = \begin{pmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{pmatrix}$$

Cette quatrième valeur ( $w_{objet}$ ) est la valeur d'homogénéité (cf. Valeur d'homogénéité), elle est fixée à 1 pour ce système de référence.

### E.5.2 Valeur d'homogénéité

Pour prendre en compte le fait que dans un espace projectif<sup>10</sup> deux droites parallèles se rencontrent à l'infini, nous devons construire un espace appelé "homogène". Définition :

$$\overrightarrow{H_{homogene}} = (x, y, z, w) \text{ tel que } \overrightarrow{E_{cartesien}} = (X, Y, Z) = \left( \frac{X}{w}, \frac{Y}{w}, \frac{Z}{w} \right)$$

10. cf. définition en annexe G.



Nous pouvons remarquer, que les coordonnées cartésiennes sont devenues invariantes si on multiplie vecteur H par un scalaire différent de zéro.

Dès lors, notre espace cartésien est devenu homogène et "w" est sa valeur d'homogénéité.

### E.5.3 ModelView Matrix

Les coordonnées de vue résultent de l'opération suivante :

$$\overrightarrow{V_{eye}} = \begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{pmatrix} = M_{modelView} \cdot \overrightarrow{V_{obj}} = M_{view} \cdot M_{model} \cdot \overrightarrow{V_{obj}}$$

#### E.5.3.1 Matrice du modèle

Cette matrice est la résultante du produit de l'ensemble des matrices de transformation appliqué aux coordonnées initiales du vertex.

Les transformations  $M_i$  sont ici du type rotation, translation, addition, soustraction,...

$$M_{model} = \prod_{i=1}^n M_i$$

#### E.5.3.2 Matrice de la vue

Cette matrice transforme les coordonnées de manière à simuler un déplacement du point de vue. En réalité, c'est l'objet graphique que l'on déplace dans le sens inverse du mouvement à simuler.

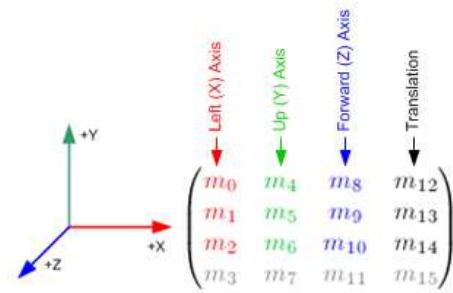


FIGURE E.4 – axes du système de référence "view" et GL\_modelView

[http://www.songho.ca/opengl/gl\\_transform.html](http://www.songho.ca/opengl/gl_transform.html)

Cette image représente la base de l'espace de dimension trois d'un point de vue quelconque

$$\overrightarrow{B_{newView}} = (\overrightarrow{B_{newX}}, \overrightarrow{B_{newY}}, \overrightarrow{B_{newZ}}) = ((m_0, m_1, m_2), (m_4, m_5, m_6), (m_8, m_9, m_{10}))$$

vis-à-vis de la base d'origine

$$\overrightarrow{B_{iniView}} = (\overrightarrow{B_{iniX}}, \overrightarrow{B_{iniY}}, \overrightarrow{B_{iniZ}}) = ((1, 0, 0), (0, 1, 0), (0, 0, 1))$$

Mais aussi, le vecteur de translation que doit subir le centre du système de référence initial (position du point de vue vis-à-vis de celui d'origine).

$$\overrightarrow{V_{translation}} = \begin{pmatrix} m_{12} \\ m_{13} \\ m_{14} \end{pmatrix}$$

Pour conclure,  $m_{15}$  est la valeur d'homogénéité et dans le cas où  $(m_3, m_7, m_{11}) = (0, 0, 0)$  et que  $m_{15} = 1$ , il est alors probable que cette matrice porte un nom ambigu car celle-ci s'apparenterait à une la matrice de vue et non comme son nom l'indique une combinaison entre la matrice du modèle et celle de la vue!

### E.5.4 Projection Matrix

Les coordonnées de clip résultent de l'opération suivante :

$$\overrightarrow{V_{clip}} = \begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix} = M_{projection} \cdot \overrightarrow{V_{eye}}$$

OpenGL prend en charge via ses fonctions deux types de projections : perspective et orthographique. Ces deux types embrassent une région de l'espace à restituer, appelée "viewing frustum", différente et l'opération qui consiste à ne prendre en compte que la partie du monde situé à l'intérieur de cet espace, s'appelle "frustum culling".

Cette opération se base sur la relation suivante pour sélectionner les sommets à écarter du rendu final :

$-w_{clip} < x_{clip}, y_{clip}, z_{clip} < w_{clip}$  avec  $w_{clip} =$

$w_{eye}$  dans le cas d'une projection orthographique.

$-z_{eye}$  avec  $z_{eye} < 0$  dans le cas d'une projection perspective.

#### E.5.4.1 Opération de projection

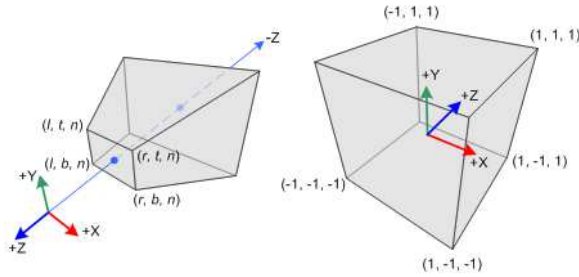


FIGURE E.5 – "Viewing frustum" de la projection perspective dans l'espace vue et cube de coordonnées "ndc"

[http://www.songho.ca/opengl/gl\\_projectionmatrix.html](http://www.songho.ca/opengl/gl_projectionmatrix.html)

Légende : r = right, l = left, n = near, b = bottom, t = top.

En ce qui concerne la matrice de projection, elle peut se déduire par cette succession d'étapes :

1. La projection des points du "viewing frustum" sur plan le plus proche du point de vue pour exprimer les coordonnées de vue en fonction des coordonnées (r,l,n et f= le plan éloigné du point de vue) définissant le "viewing frustum" au sein de ce système.

$$\overrightarrow{V_{near eye}} = M_{near projection} \cdot \overrightarrow{V_{eye}}$$

;

2. L'utilisation des relations linéaires suivantes pour exprimer les coordonnées "ndc" en fonction des coordonnées de vue :

$$\begin{aligned} [l,r] &=> [-1, 1] \text{ pour } x_{ndc} \\ [b,t] &=> [-1, 1] \text{ pour } y_{ndc} \\ [n,f] &=> [-1, 1] \text{ pour } z_{ndc} \end{aligned}$$

3. Et enfin, on utilise la définition de la relation entre les coordonnées "clip" et "ndc" pour en déduire la matrice de projection.

### E.5.5 Normalized Device Coordinates (ndc)

Les coordonnées normalisées sont définies par l'opération suivante :

$$\overrightarrow{V_{ndc}} = \begin{pmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{pmatrix} = \begin{pmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{pmatrix}$$

Cette division est appelée "perspective division".

## E.5.6 Window Coordinates

Sous OpenGL, la zone de rendu est définie par les méthodes suivantes :

**glViewport(x,y,w,h) :** définit le rectangle sur lequel le rendu apparaîtra et initialise les transformations affines nécessaires au passage des coordonnées "ndc" aux coordonnées de type écran.

$(x, h)$  correspondent aux coordonnées du coin supérieur gauche  
 $(w = largeur, h = hauteur)$  celles du coin inférieur droit.

**glDepthRange(n,f) :** définit la valeur de  $z_{win}$  en initialisant la correspondance entre les plans du "viewing frustum" et les coordonnées "win" :

$n = near$  pour le plan proche du "viewing frustum" et les coordonnées "win"  
 $f = far$  pour le plan éloigné du "viewing frustum"

Les coordonnées d'écran résultent de l'opération suivante :

$$\overrightarrow{V_{win}} = \begin{pmatrix} x_{win} \\ y_{win} \\ z_{win} \end{pmatrix} = \begin{pmatrix} \frac{w}{2} \cdot x_{ndc} + (x + \frac{w}{2}) \\ \frac{h}{2} \cdot y_{ndc} + (y + \frac{h}{2}) \\ \frac{f-n}{2} \cdot z_{ndc} + \frac{f+n}{2} \end{pmatrix}$$

## E.6 Le langage GLSL

L'OpenGL Shading Language (GLSL) est un langage de programmation pour les shaders.

Ci-dessous, on trouvera seulement les définitions qui concernent les éléments utilisés au sein de cette application.

**precision :**

précise le taux de précision avec laquelle le GPU<sup>11</sup> utilisera le type de données introduit.

**attribute :**

ce qualificatif est utilisé pour déclarer des variables qui sont partagées entre un vertex shader et l'environnement OpenGL ES.

**varying :**

qualificatif qui permet de faire transiter une variable définie pour le vertex shader vers le fragment shader.

**uniform :**

ce qualificatif est utilisé pour déclarer des variables globale qui sont partagées entre un shader et l'environnement OpenGL ES.

Le qualificatif "Global" signifie que cette variable ne change pas d'une instance de shader à une autre au sein d'un appel de rendu.

**sampler2D :**

variable utilisée pour représenter une texture en deux dimension.

**texture2D :**

variable responsable de la transmission des coordonnées de textures.

---

11. cf. définition en annexe G.

# Annexe F

## Les repères de coordonnées

### F.1 Le repère géographique

La position "P" d'un lieu géographique sur la Terre est déterminée à l'aide de deux valeurs :

**La latitude  $\phi$  :**

angle mesuré dans le plan que forme le méridien du lieu, entre le plan de l'équateur et la verticale du lieu. Cet angle vaut  $0^\circ$  à l'équateur et varie de manière positive vers le nord et négative vers le sud pour atteindre  $\pm 90^\circ$  aux pôles.

**La longitude  $\lambda$  :**

angle mesuré dans le plan équatorial, entre le plan que forme le méridien de Greenwich et que forme le méridien du lieu. Cet angle varie de  $0^\circ$  à  $\pm 180^\circ$ , positivement vers l'ouest et négativement vers l'est.

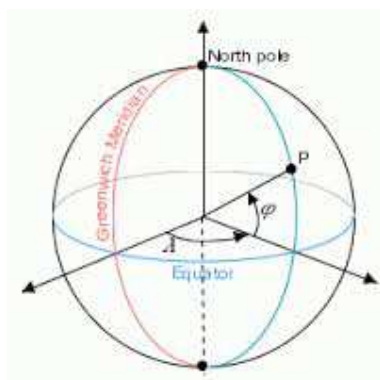


FIGURE F.1 – Repère géographique.

**Légende** ==>  $\lambda$  : longitude. |  $\phi$  : latitude. |

**Image** : <http://pic.dhe.ibm.com/>

### F.2 Le repère azimutal

Ce repère est utilisé pour déterminer la position d'un corps dans le ciel par rapport au méridien et à l'horizon du lieu d'observation.

Les coordonnées du corps sont déterminées par :

**L'azimut "a" :**

angle mesuré dans le plan horizontal, entre le méridien du lieu (le sud) et le demi-méridien du corps. Cet angle varie de  $0^\circ$  à  $360^\circ$  positivement vers l'ouest.

**La hauteur "h" :**

angle mesuré dans le plan contenant le demi-méridien du corps, entre le plan horizontal et sa position. Cet angle varie de  $0^\circ$  à  $90^\circ$  (zénith).

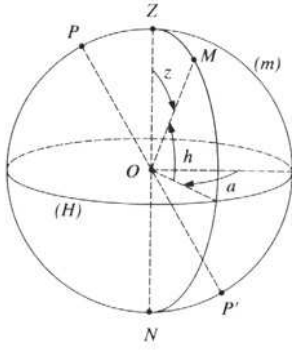


FIGURE F.2 – Repère azimutal.

**Légende** => a : azimut. | h : hauteur. | z : colatitute ( $90^\circ$  - latitude). |

**Image** : <http://www.imcce.fr/langues/fr/grandpublic/systeme/promenade/pages3/316.html>

En ce qui concerne la détermination de la position d'un corps céleste, ce système n'est pas approprié car l'azimut et la hauteur d'un corps céleste varient en fonction du temps (rotation de la Terre).

### F.3 Le repère horaire

Ce système utilise l'axe des pôles comme direction de référence afin de s'affranchir de la variation due au mouvement diurne<sup>1</sup> sur l'une de ses coordonnées (la déclinaison).

L'origine de ce repère est déterminé par le plan de l'équateur terrestre et le méridien du lieu d'observation.

Les coordonnées d'un corps céleste sont déterminées par :

**L'angle horaire "H" :**

angle mesuré, dans le plan équatorial, entre le méridien du lieu d'observation (sud) et le cercle horaire du corps céleste. Cet angle varie de 0 et 24 heures positivement vers l'ouest.

**La déclinaison  $\delta$  :**

angle mesuré dans le plan contenant le demi-méridien du corps céleste, entre le plan équatorial et sa position. Cet angle vaut  $0^\circ$  à l'équateur et varie de manière positive vers le nord et négative vers le sud pour atteindre  $\pm 90^\circ$  aux pôles.

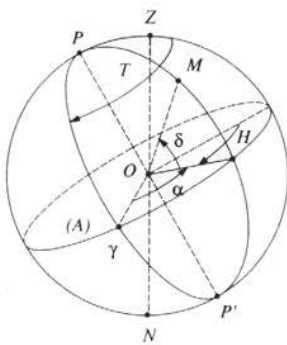


FIGURE F.3 – Repère horaire.

**Légende** =>  $\gamma$  : le point vernal. |  $\alpha$  : ascension-droite. |  $\delta$  : déclinaison. | H : angle horaire. | M : une étoile. | T : temps sidéral local (angle séparant le méridien du lieu du point vernal  $\gamma$ ). |

**Image** : <http://www.imcce.fr/langues/fr/grandpublic/systeme/promenade/pages3/316.html>

1. cf. définition en annexe G

## F.4 Le repère équatorial

Ce repère sert à déterminer la position d'un corps céleste car il ne dépend pas du mouvement diurne de la Terre.

L'origine de ce repère est déterminée par le plan de l'équateur terrestre et le point vernal.

### Le point vernal :

point d'intersection entre le plan de l'équateur et celui de l'écliptique (plan orbital de la Terre autour du Soleil) au moment de l'équinoxe de printemps (noeud ascendant).

Etant donné que ce point varie en fonction des mouvements de précession et de nutation de l'axe de rotation de la Terre, les coordonnées des étoiles seront données en précisant le moment où elles ont été calculés.

Les coordonnées d'un corps céleste sont déterminées par :

#### L ascension droite $\alpha$ :

angle de type horaire mesuré, dans le plan équatorial, entre le point vernal et le cercle horaire du corps céleste. Cet angle varie de 0 et 24 heures positivement vers l'est.

#### La déclinaison $\delta$ :

angle mesuré dans le plan contenant le demi-méridien du corps céleste, entre le plan équatorial et sa position. Cet angle vaut  $0^\circ$  à l'équateur et varie de manière positive vers le nord et négative vers le sud pour atteindre  $\pm 90^\circ$  aux pôles.

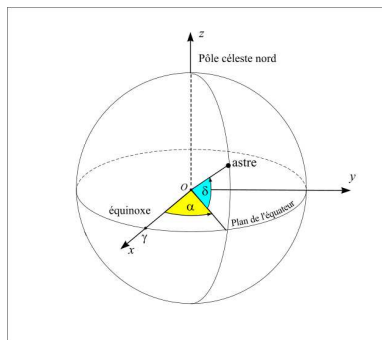


FIGURE F.4 – Repère équatorial.

**Légende** =>  $\alpha$  : ascension-droite. |  $\delta$  : déclinaison. |

**Image** : <http://www.imcce.fr/langues/fr/grandpublic/systeme/promenade/pages3/316.html>

1. cf. définition en annexe G 1. pour exemple, voir la définition de "J2000" en annexe G 1. angle exprimé en heures (1h = 15°), minutes d'arc (1/60 °) et secondes d'arc (1/3600 °) 1. cf. définition en annexe G

# Annexe G

## Définitions

Certaines des définitions ci-dessous proviennent du site internet de "Larousse"<sup>1</sup>, Wikipédia<sup>2</sup> et de Institut de mécanique céleste et de calcul des éphémérides (IMCCE)<sup>3</sup>.

**Android :** système d'exploitation open source utilisant le noyau Linux et adapté aux environnements mobiles. (*Définition Wikipédia.*)

**Année julienne :** Unité auxiliaire de temps définie comme étant égale à 365.25 jours. (*Définition IMCCE.*)

**Année-lumière :** L'année-lumière est une unité de mesure de distance (et non de temps). L'Union astronomique internationale la définit comme la distance parcourue par un photon dans le vide, en une année julienne (soit 365,25 jours, ou 31 557 600 secondes). La vitesse de la lumière dans le vide étant une constante fixée à 299 792 458 m/s, une année-lumière est exactement égale à 9 460 730 472 580,8 km. (*Définition Wikipédia.*)

**Buffer :** mémoire tampon, une zone de mémoire virtuelle ou de disque dur utilisée pour stocker temporairement des données, notamment entre deux processus ou deux pièces d'équipement ne fonctionnant pas à la même vitesse. (*Définition Wikipédia.*)

**Cercle horaire :** Le cercle horaire d'un astre est un grand cercle de la sphère céleste obtenu par l'intersection de la sphère et d'un plan qui passe par l'astre considéré et l'axe des pôles célestes. (*Définition Wikipédia.*)

**CPU :** Le processeur (ou CPU de l'anglais Central Processing Unit, « Unité centrale de traitement ») est le composant de l'ordinateur qui exécute les instructions machine des programmes informatiques. (*Définition Wikipédia.*)

**Date julienne (DJ) :** durée écoulée depuis le 1 janvier - 4712 à 12 h, origine de la période julienne. On l'exprime en jour et fraction de jour. Pour un usage rigoureux, on doit préciser l'échelle de temps utilisée (TU, TT, TE, etc.). (*Définition IMCCE.*)

**EGL :** est une Interface de programmation (API) du Khronos Group, faisant le lien entre ses API de rendu comme OpenGL ES, et le système de fenêtrage du système d'exploitation sous-jacent.

EGL prend en charge la gestion des contextes graphiques, les relations entre les surfaces de dessin et les tampons mémoire, ainsi que la synchronisation du rendu. EGL permet ainsi la gestion efficace de rendu graphique mixte (2D et 3D) accéléré matériellement. (*Définition Wikipédia.*)

**Espace projectif :** En mathématiques, un espace projectif est une construction fondamentale qui permet d'homogénéiser un espace vectoriel, autrement dit d'oublier les proportionnalités pour ne plus considérer que des directions. L'utilisation d'espaces projectifs rend rigoureuse les notions de droite à l'infini, qui est l'ensemble des points à l'infini où les droites parallèles se rencontrent, et de plan à l'infini (en) pour trois dimensions, et les généralisent. On peut choisir arbitrairement un plan dans l'espace projectif comme plan à l'infini. On obtient un espace projectif en ajoutant une coordonnée supplémentaire à celle d'un espace ordinaire ; exemple : trois pour un espace à deux dimensions ; quatre pour un espace à trois dimensions ;

1. cf. bibliographie [13]. 2. cf. bibliographie [19]. 3. cf. bibliographie [6].

etc. Ainsi le point de coordonnées  $(x,y,z)$  en 3D aura en représentation projective les coordonnées  $(x,y,z,1)$ . Les points à l'infini ont pour coordonnées  $(x,y,z,0)$  avec  $x,y,z$  non tous nuls ; par exemple celui de l'axe des  $x$  a pour coordonnées  $(1,0,0,0)$ . (*Définition Wikipédia.*)

**Fragment :** Un fragment est un ensemble d'état (bit) qui est utilisé pour calculer les données finales pour un pixel (ou un échantillon si multi-échantillonnage est activé) dans le framebuffer de sortie. L'état d'un fragment comprend sa position dans l'espace écran, la couverture de l'échantillon si multi-échantillonnage est activé, et une liste de données arbitraires qui était sortie du vertex shader de géométrie si celui-ci a fait partie du pipeline de rendu. (*Définition OpenGL.org.*)

**Framebuffer :** Un framebuffer (soit en français tampon de trame ou mémoire d'image) est un équipement de sortie vidéo qui commande un affichage vidéo à partir d'un tampon mémoire contenant une trame complète de données. Les informations dans le tampon mémoire sont en général les valeurs des composantes de couleurs pour chaque pixel (chaque point à afficher sur l'écran). Les valeurs sont habituellement mémorisées sous les formats 1-bit monochrome, 4-bit palettisées, 8-bit palettisées, 16-bit highcolor ou 24-bit truecolor (vraies couleurs). Un canal alpha (alpha channel) est parfois utilisé pour l'information de transparence associée à chaque pixel. (*Définition Wikipédia.*)

**Global Positioning System :** système américain de navigation et de localisation par satellite. (*Définition Larousse.*)

**GPU :** Un processeur graphique, ou GPU (de l'anglais Graphics Processing Unit), est un circuit intégré la plupart du temps présent sur une carte graphique (mais pouvant aussi être intégrée sur une carte-mère ou dans un CPU) et assurant les fonctions de calcul de l'affichage. Un processeur graphique a généralement une structure hautement parallèle (voir accélération matérielle) qui le rend efficace pour une large palette de tâches graphiques comme le rendu 3D en OpenGL. (*Définition Wikipédia.*)

**Environnement de développement (Integrated Development Environment) :** ensemble d'outils (éditeur de texte, compilateur, débogueur) pour augmenter la productivité des programmeurs qui développent des logiciels. (*Définition Wikipédia.*)

**J2000 = Origine des temps (ou époque standard) :** en 1984 l'origine des temps a été fixée au 1 janvier 2000 à 12 heures de l'échelle de temps considérée. Elle correspond à la date julienne 2 451 545.0 et est désignée par J2000.0 ou J2000. Par définition le début d'une année julienne est séparé de l'époque standard par un nombre entier d'années juliennes. (*Définition IMCCE.*)

**Magnitude apparente :** la magnitude apparente est une mesure de l'irradiance d'un objet céleste observé depuis la Terre. La magnitude augmente d'une unité lorsque l'irradiance est divisée par 2,51. (*Définition Wikipédia.*)

**Méridien :** Le méridien, ou plan méridien, d'un lieu est un grand cercle de la sphère céleste passant par le pôle céleste, le zénith et le nadir du lieu. À midi solaire, le Soleil est dans le plan méridien. (*Définition Wikipédia.*)

**Mouvement diurne :** En astronomie, le mouvement diurne est le mouvement apparent de rotation des astres dans le ciel autour d'un pôle céleste. (*Définition Wikipédia.*)

**Précession-nutation :** Déplacement au cours du temps du plan de l'équateur et du plan de l'écliptique, par rapport à un système de référence inertiel, dû aux actions gravitationnelles de la Lune, du Soleil et des planètes. La représentation mathématique de ce déplacement comporte des termes séculaires, des séries périodiques et des séries de Poisson. Conventionnellement on appelle précession l'ensemble des termes séculaires et nutation l'ensemble des séries périodiques et des séries de Poisson. (*Définition IMCCE.*)

**Système d'exploitation (Operating System) :** ensemble de programmes qui dirige l'utilisation des capacités d'un ordinateur. (*Définition Wikipédia.*)

**Radian :** un radian est une unité d'angle qui vaut environ  $57,3^\circ$  ( $360^\circ/2\pi$ ). Cet angle de  $57,3^\circ$  intercepte un arc de longueur égale au rayon. (*Définition Wikipédia.*)



**Shader** : un shader (le mot est issu du verbe anglais to shade pris dans le sens de « nuancer ») est un programme informatique, utilisé en image de synthèse, pour paramétrer une partie du processus de rendu réalisé par une carte graphique ou un moteur de rendu logiciel. (*Définition Wikipédia.*)

**texel** : En infographie, le texel est le plus petit élément d'une texture appliquée à une surface. Texel vient de l'anglais texture element.

La différence entre le pixel et le texel est que ce dernier peut occuper la place de plusieurs pixels (de l'image rendue à l'écran) ou bien être de taille inférieure au pixel (dans ce cas, il n'est pas forcément visible mais il existe toujours).

Il peut aussi être mélangé à un autre texel, suivant la distance de l'objet par rapport à la caméra. (*Définition Wikipédia.*)

**Thread** : un thread ou fil (d'exécution) est similaire à un processus car tous deux représentent l'exécution d'un ensemble d'instructions du langage machine d'un processeur. (*Définition Wikipédia.*)

**Transformation conforme** : En mathématiques, une transformation conforme est une bijection qui conserve localement les angles, c'est-à-dire qui se comporte au voisinage de chaque point où elle est définie presque comme une similitude. (*Définition Wikipédia.*)

**Transformation équivalente** : Transformation qui conserve les aires. (*Définition Wikipédia.*)

**Kit de développement (Software Development Kit)** : ensemble d'outils permettant aux développeurs de créer des applications de type défini (par exemple pour Android). (*Définition Wikipédia.*)

**Base de données relationnelle** : En informatique, une base de données relationnelle est un stock d'informations décomposées et organisées dans des matrices appelées relations ou tables conformément au modèle de données relationnel. (*Définition Wikipédia.*)

**SQL** : SQL (Structured Query Language) est un langage informatique normalisé servant à exploiter des bases de données relationnelles. (*Définition Wikipédia.*)

**Trame** : Ensemble des lignes horizontales explorées au cours d'un balayage vertical unique d'une image.

Dans les systèmes actuels, pour éviter le papillotement, chaque image comprend deux trames, l'une pour les lignes paires, l'autre pour les lignes impaires. (*Définition Larousse.*)

**Unité astronomique (ua)** : historiquement basée sur la distance entre la Terre et le Soleil, l'unité astronomique est définie comme valant exactement 149 597 870 700 m (*Définition Wikipédia.*)

**Temps universel (UT)** : échelle de temps étroitement liée à la rotation diurne de la Terre qui a longtemps été à la base des temps légaux. TU est défini par une relation mathématique donnant l'expression du temps sidéral en fonction du Temps universel. On peut donc déterminer TU à partir d'observations d'étoiles (passage d'étoiles au méridien, par exemple). Le Temps universel ainsi obtenu est rapporté à un pôle fixe sur la Terre et est noté UT0. Le Temps universel rapporté au pôle céleste des éphémérides CEP s'obtient en s'affranchissant du mouvement du pôle et est noté UT1. Depuis 1984 l'échelle de temps légale n'est plus basée sur le Temps universel mais sur le Temps universel coordonné UTC. (*Définition IMCCE.*)

**Temps universel coordonné (UTC)** : échelle de temps diffusée par les signaux horaires et utilisée comme base des temps légaux. C'est, en fait le Temps atomique international TAI décalé d'un nombre entier de secondes. Ce nombre est modifié régulièrement de telle sorte que la différence entre UTC et le Temps universel UT1 n'excède pas 0.9 s en valeur absolue. (L'heure moyenne de Greenwich (GMT) est définie depuis 1972 par UTC.) (*Définition IMCCE.*)

**Vertex** : un point ou sommet d'un polygone sous la forme de coordonnées 2D ou 3D. Un ensemble de vertex est appelé vertice.

**La Voie Lactée :** La Voie lactée est la galaxie dans laquelle se situe le Système solaire. Elle est partiellement visible dans de bonnes conditions d'observation (absence de pollution lumineuse), notamment sous les tropiques, sous la forme d'une bande plus claire dans le ciel nocturne. (*Définition Wikipédia.*)

# Bibliographie

- [1] Song Ho Ahn. <http://www.songho.ca/>.
- [2] AndroidDevelopers. <http://developer.android.com/index.html>.
- [3] International astronomical union. <https://www.iau.org/public/themes/constellations/>.
- [4] Meeus J ; Savoie D. *Emploi des éphémérides de position*. Société astronomique de France (SAF), 1997. Isbn : ?
- [5] Centre de données astronomiques de Strasbourg. <http://cds.u-strasbg.fr/>.
- [6] L'Institut de mécanique céleste et de calcul des éphémérides.  
"<http://www.imcce.fr/fr/grandpublic/glossaire.php>.
- [7] Pascoli Gianni. *Astronomie fondamentale. Astronomie de position et mécanique céleste*. Dunod, 2000. Isbn : 2-10-005214-4.
- [8] Allen Grant. *L'art du développement Android*. Pearson, 2012. Isbn : 978-2-7440-2557-0.
- [9] Brigitte Hergigny. Le processus unifié (rup). Cours => Analyse : principes et méthodes (ANPM) , EPFC Enseignement de Promotion et de Formation Continue Bruxelles.
- [10] Meeus Jean. *Calculs astronomiques à l'usage des amateurs*. Société astronomique de France (SAF), 2014. Isbn : 978-2-901730-05-7.
- [11] Ochsenbein F.; Halbwachs J.L. Catalogue of the brightest stars (/catalog or /names).  
<http://vizier.u-strasbg.fr/viz-bin/VizieR?-source=V%2F53A>.
- [12] Hoffleit D.; Warren W. H. Jr. The bright star catalogue (bsc).  
<http://vizier.u-strasbg.fr/viz-bin/VizieR?-source=V%2F50>.
- [13] Larousse. <http://www.larousse.fr/dictionnaires/francais>.
- [14] Kittel C; Knight D; Ruderman M. *BERKELEY Cours de physique, 1. mécanique*. Dunod, 2001. Isbn : 2 10 005517 8.
- [15] OpenGL.org. [https://www.opengl.org/wiki/Main\\_Page](https://www.opengl.org/wiki/Main_Page).
- [16] Oracle. <http://www.oracle.com/fr/index.html>.
- [17] Gauthier Picard. Conduite et gestion de projets informatiques.  
<http://www.emse.fr/~picard/cours/2A/gp/GP-Introduction.pdf>, 2009. École Nationale Supérieure des Mines de Saint-Étienne France.
- [18] Gauthier Picard. Processus unifié.  
<http://www.emse.fr/~picard/cours/2A/gp/GP-UnifiedProcess.pdf>, 2009. École Nationale Supérieure des Mines de Saint-Étienne France.
- [19] Wikipédia. <http://fr.wikipedia.org/>.