Rapport de Stage

Développement d'Alipad

Version simplifiée pour Android du mode Allsky d'Aladin, l'Atlas 3D du CDS

Stagiaire : Vincent Kaestle
Maîtres de stage : André Schaaff, Pierre Fernique
Enseignant tuteur : Dominique Colnet

Année universitaire 2010-2011

Centre de Données Astronomiques de Strasbourg
Observatoire Astronomique de Strasbourg
11, rue de l'Université
67000 Strasbourg

Université Nancy 2
IUT Nancy-Charlemagne
2, ter boulevard Charlemagne
54052 Nancy

Rapport de Stage

Développement d'Alipad

Version simplifiée pour Android du mode Allsky d'Aladin,
l'Atlas 3D du CDS

Stagiaire : Vincent Kaestle
Maîtres de stage : André Schaaff, Pierre Fernique
Enseignant tuteur : Dominique Colnet

Année universitaire 2010-2011
S'est déroulé du 11 avril au 17 juin 20011

Centre de Données Astronomiques de Strasbourg
Observatoire Astronomique de Strasbourg
11, rue de l'Université
67000 Strasbourg

Université Nancy 2
IUT Nancy-Charlemagne
2, ter boulevard Charlemagne
54052 Nancy

Remerciements

Je tiens tout d'abord à remercier André Schaaff, ingénieur de recherche au Centre de Données astronomiques de Strasbourg, d'avoir bien voulu me faire confiance en m'intégrant à son équipe de recherche et de m'avoir encadré tout au long de mon stage.

Je remercie également Pierre Fernique, ingénieur de recherche au Centre de Données astronomiques de Strasbourg, de m'avoir conseillé et encadré durant mon stage.

Je remercie Dominique Colnet pour avoir été mon responsable enseignant de l'IUT Nancy Charlemagne et pour nous avoir rendu visite.

Je remercie aussi les informaticiens, astronomes et documentalistes qui m'ont permis de découvrir le contexte dans lequel je travaillais, à travers différentes présentations des services du CDS ainsi que de leur travail au sein de l'Observatoire.

Enfin je remercie toutes les personnes travaillant à l'Observatoire astronomique de Strasbourg pour leur très bon accueil en particulier André Schaaff, et Sébastien Derrière, astronome, qui ont partagé leur bureau avec moi pendant ces quelques semaines.

Table des matières

Introduction

Pour clôturer mon DUT informatique, je devais effectuer un stage de 10 semaines. Je cherchais un stage portant sur le développement d'une application, et ce n'était pas forcément ce qu'il y avait de plus courant au niveau bac+2. Généralement on nous propose plutôt de développer un site Web, ou de gérer une base de données, ou encore d'installer un réseau informatique. J'avais donc transmis mon cv auprès de filiales de Dassault et de la Française des Jeux, mais bien souvent, une formation d'ingénieur était requise.

Ensuite en regardant les annonces publiées sur le réseau de l'IUT, je suis tombé sur celle proposée par André Schaaff. Elle concernait le développement d'application de type atlas interactif sur smartphone et divers appareils tactiles. J'avais déjà développé sur Iphone durant mon projet tutoré et j'avais déjà utilisé des librairies de graphisme 3D telles qu'OpenGL. Ces compétences avaient l'air d'être adaptées à ce genre de stage. J'ai donc candidaté à l'annonce en prenant contact avec André Schaaff. Après une visite de l'Observatoire fin janvier, le sujet du stage fut établi.

Ainsi, dans un premier temps, je vous présenterai l'Observatoire Astronomique de Strasbourg avec en particulier les services du CDS. Et ensuite je reviendrai plus en détail sur l'objet de mon stage et comment il s'est déroulé. Nous verrons notamment les prototypes réalisées afin de mieux s'attaquer au développement de l'application.

Chapitre 1

Présentation de l'Observatoire

1.1 L'Observatoire Astronomique de Strasbourg

1.1.1 Présentation générale

L'Observatoire Astronomique de Strasbourg est un Observatoire des Sciences de l'Univers, et une Unité Mixte de Recherche du CNRS [1] et des universités de Strasbourg.

Il a été fondé en 1881 et est situé sur le campus historique de l'université de Strasbourg. Il est constitué de trois bâtiments à côté desquels se trouve le planétarium destiné au grand public.

L'Observatoire abrite des activités de recherche et d'enseignement, les services du Centre de Données de Strasbourg (CDS, cf détails au 1.2), et héberge des activités de diffusion de la culture avec le Planétarium.

1. Centre National de la Recherche Scientifique.

1.1.2 Les domaines de recherche

Les équipes de recherche de l'Observatoire sont au nombre de quatre : les Hautes Energies, les Etoiles et Système Stellaires, les Galaxies et le Centre de Données astronomiques. Voici un court résumé du travail de ces équipes.

Hautes Energies

Les travaux de l'équipe Hautes Energies portent sur l'astrophysique des hautes énergies, plus particulièrement sur l'étude observationnelle et théorique des objets stellaires jeunes et des objets compacts dans notre Galaxie mais aussi dans d'autres galaxies.

Les chercheurs de l'équipe Hautes Energies utilisent les instruments performants d'observation en rayons X à bord des satellites Chandra [2] et XMM-Newton [3], ainsi que les télescopes au sol avec leurs instruments de haute qualité mis à la disposition de la communauté française.

L'équipe Hautes Énergies est partie prenante du Survey Science Center (SSC) du satellite XMM-Newton, depuis sa création en 1996. Elle y joue un rôle essentiel, et accomplit dans ce cadre une activité de service.

Etoiles et systèmes stellaires

Les recherches menées par l'équipe Etoiles et systèmes stellaires recouvrent un domaine étendu, incluant les étoiles, les milieux interstellaires, la Galaxie et les galaxies proches. Dans le domaine stellaire, on retrouve la plupart des enjeux scientifiques recensés par le PNPS [4], et l'ensemble des objectifs prioritaires.

Ces thèmes ont été abordés en faisant appel à des données nouvelles, comme les catalogues Hipparcos et Tycho [5], ou des observations réalisées avec des instruments hautement performants.

C'est ainsi qu'a pris corps une large palette thématique, dont les acteurs interviennent parfois sur des sujets très différents, allant de l'objet individuel aux structures à grandes échelles.

2. Télescope à rayon X lancé en 1999 par la navette spatiale Columbia.
3. Satellite artificiel d'observation des rayons X lancé par l'Agence spatiale européenne (ESA) en 1999.
4. Programme National de Physique Stellaire.
5. Catalogues contenant des milliers d'étoiles publiés par l'ESA en juin 1997.

Galaxies

Les activités de l'équipe Galaxies couvrent des problèmes variés concernant l'histoire et la formation des galaxies ainsi que l'étude des populations stellaires qui constituent ces galaxies.

Plus particulièrement, ils étudient les caractéristiques et les propriétés physiques de notre Galaxie, des galaxies du Groupe Local ainsi que celles d'autres galaxies « proches » de nous. Ils observent et analysent les populations stellaires de ces galaxies et étudient la dynamique gravitationnelle qui régit les mouvements du gaz et des étoiles. Un de leurs objectifs consiste à combiner les informations liées à l'histoire et l'évolution des populations stellaires, ainsi que celles liées à la dynamique et à la cinématique des étoiles ; ceci afin de reconstituer les événements clés de la vie de ces galaxies.

Parmi les autres thèmes de recherche, la cinématique et la dynamique gravitationnelle sont étudiées en tant que telles car ce sont des phénomènes physiques fondamentaux qui régissent le mouvement des étoiles et qui peuvent expliquer la morphologie des galaxies et leur évolution temporelle.

1.2 Le Centre de Données Astronomiques de Strasbourg

1.2.1 Présentation générale

Le Centre de Données astronomiques de Strasbourg (CDS) créé en 1972 remplit pour la communauté scientifique internationale une mission qui peut se résumer de la façon suivante : Collecter, homogénéiser, distribuer et préserver l'information astronomique pour le bénéfice de l'ensemble de la communauté scientifique internationale.

Il est composé de 30 personnes parmi les 80 personnes travaillant à l'observatoire. On compte 2 administratifs, 8 assistants ingénieur et ingénieurs d'étude documentalistes, 2 CDD ingénieurs de recherche en informatique, 8 ingénieurs d'étude et ingénieurs de recherche en informatique, 10 chercheurs et enseignants-chercheurs. Le CDS dispose de sa propre infrastructure (salles de serveurs, etc.) et tous les postes de travail sont basés sur Ubuntu 8.04 - Hardy Heron.

Les services du CDS sont largement utilisés par la communauté astronomique internationale et leur utilisation est en augmentation continue.

Ces services constituent la partie la plus visible de l'activité du CDS, et leur qualité est la base de la crédibilité de celui-ci au niveau international. Mais la mission du CDS se décline en fait sur plusieurs types d'activités, qui sont

décrites ci-dessous :
- Construction de services de référence à forte valeur ajoutée
- Définition de standards et d'outils génériques
- Participation à des projets
- Support projet
- Veille technologique, Recherche et Développement
- Diffusion des connaissances

La période 2000-2003 a vu l'émergence très rapide du concept d'Observatoire Virtuel. Le CDS joue un rôle moteur dans les projets européens (AVO, Euro VOTECH, EuroVO DCA, EuroVO AIDA, EuroVO ICE) liés à ce concept. Dans la continuité de ses activités de support aux projets prioritaires de la discipline, le CDS joue également un rôle moteur dans l'organisation de la communauté nationale (actions spécifiques OV France) pour faire face à ce défi nouveau.

1.2.2 Les principaux services du CDS

La construction de services de référence à forte valeur ajoutée constitue la base de l'activité du Centre de Données.

Le CDS développe et maintient trois services principaux qui constituent chacun des références dans le milieu de l'astronomie. Ils ont tous les trois leur propre utilité mais sont étroitement liés. Le portail du CDS s'efforce de rendre leur utilisation simple et pratique.

Simbad

Simbad est la base de données de référence pour la nomenclature et la bibliographie des objets astronomiques. En plus de 40 années de service, elle a accumulé pas moins de 5,4 millions d'objets, 15,1 millions d'identificateurs et 254 000 références bibliographiques.

En traitant des centaines de milliers de requêtes par jour, c'est aujourd'hui une référence mondiale sans équivalent.

Ce service permet, à partir du nom d'un objet astronomique, de retrouver des informations comme sa position, son type ainsi que des mesures effectuées et l'ensemble des articles où il est référencé (cf figure 1.1).

D'autres méthodes pour interroger la base sont possibles, par exemple avec les coordonnées de l'objet ou encore par ses références.

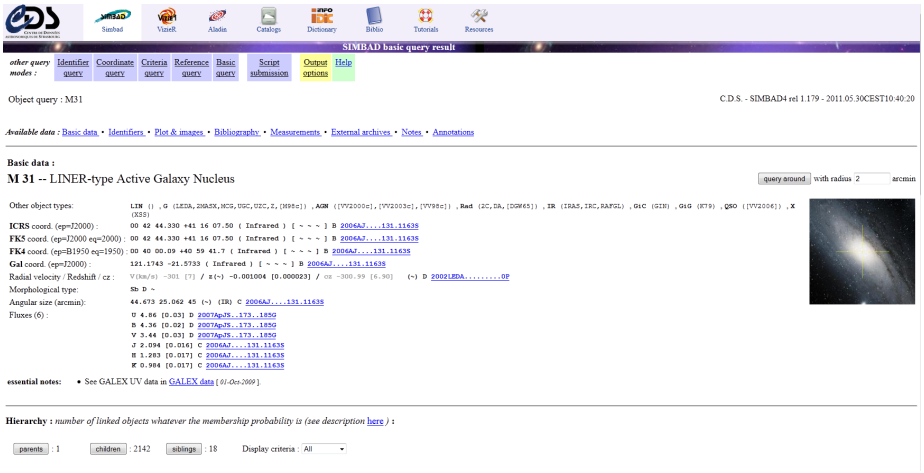

FIGURE 1.1 – Capture d'écran de Simbad

VizieR

C'est une base de données qui regroupe environ 9000 catalogues d'objets astronomiques. Ces catalogues sont en fait des tables relevées durant des observations et rentrées dans VizieR par les documentalistes.

Ce service permet à un utilisateur d'accéder de manière homogène à des données hétérogènes de catalogues, de les croiser et de les exporter sous différents formats (cf figure 1.2).

Les interrogations sur la base de catalogues peuvent porter sur différents critères comme la longueur d'onde avec laquelle les objets ont étés observés ou encore la mission correspondante.

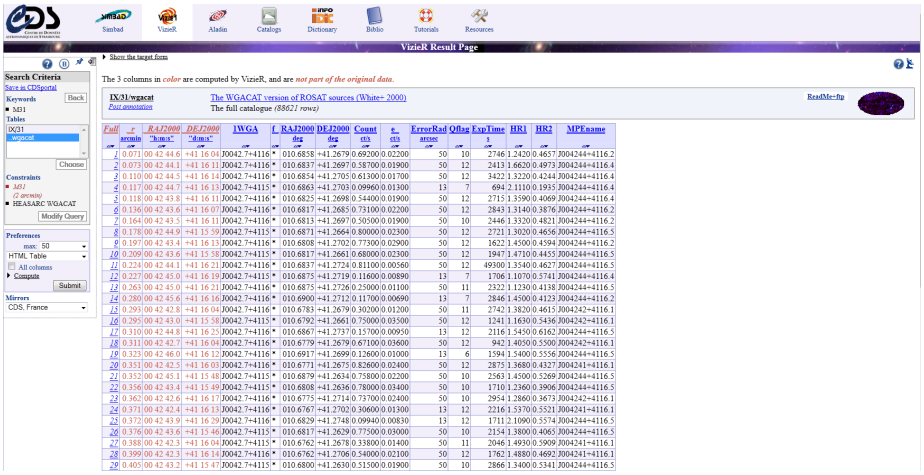

FIGURE 1.2 – Capture d'écran de VizieR

Aladin

Aladin est un atlas du ciel interactif. C'est le service le plus récent, il date de 1999 et est développé et maintenu par Pierre Fernique, Thomas Boch et François Bonnarel.

Ce service permet de visualiser des images digitalisées de n'importe quelle partie du ciel, d'y superposer des objets issus de catalogues astronomiques et, interactivement, d'accéder aux informations relatives (cf figure 1.3).

Les catalogues d'images proviennent des bases de données du CDS mais également d'autres catalogues internationaux.

Le logiciel entièrement écrit en Java est disponible sous forme d'applet sur le site du CDS mais également en version Standalone [6] en téléchargement libre. Le code source est quant à lui distribué sous licence GPL [7].

6. Une version Standalone est une application à part entière capable de fonctionner indépendamment de toute autre application.

7. General Public License ; Libre mais oblige celui qui publie un logiciel utilisant ce code source a être également sous licence GPL.

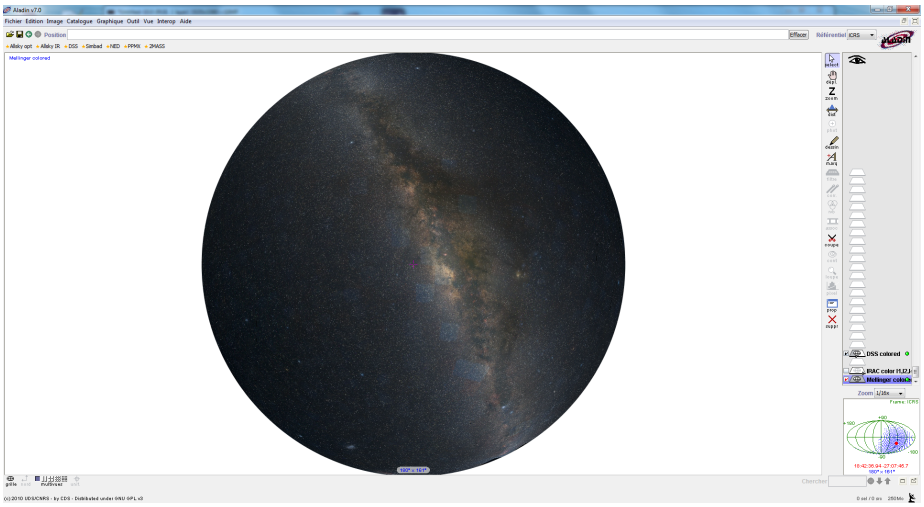

FIGURE 1.3 – Capture d'écran d'Aladin

1.3 International Virtual Observatory Alliance

1.3.1 Présentation

L'IVOA est une organisation internationale ayant pour but de faciliter les échanges internationaux et la collaboration nécessaire à l'élaboration de standards d'interopérabilité et au déploiement d'outils, de systèmes et de structures organisés nécessaires pour permettre une utilisation internationale des archives astronomiques (images, catalogues, etc..). Telle est la mission confiée à l'IVOA lors de sa création en juin 2002.

La participation à l'IVOA est libre et à l'heure actuelle on compte une vingtaine de projets parmi lesquels ont peut citer l'Arménie, l'Australie, le Canada, la Chine, l'Europe, la France, l'Allemagne, la Hongrie, l'Inde, le Brésil, l'Italie, la Corée, le Japon, la Russie, l'Espagne, le Royaume-Uni et les Etats-Unis. Des discussions sont en cours avec de nombreux autres candidats souhaitant participer à cet effort.

L'ESA [8] et l'ESO [9] participent activement à cette internationalisation et standardisation des échanges.

8. Agence Spatiale Européenne créée en 1975
9. Organisation Européenne pour la recherche en Astronomie fondée en 1962.

1.3.2 Le CDS au sein de l'IVOA

En 2009, le Centre de Données est reconnu comme une « très grande infrastructure de Recherche » par l'état. Il joue un rôle, depuis une quarantaine d'années, dans les collaborations internationales. Il est reconnu par la communauté des astronomes du monde entier et on peut dire qu'il a un rôle très important au sein de l'IVOA. Il participe activement à ses travaux.

L'organisation au sein de l'IVOA se fait par groupes de travail, chacun constitué de membres de plusieurs partenaires dont le but est de discuter des standards à mettre en place. Le CDS joue un rôle primordial car il participe à la plupart de ces groupes de travail et en dirige ou co-dirige certains :
– Sebastien Derriere pour le groupe de travail sémantique qui s'occupe notamment de l'élaboration et la mise en place des UCD [10].
– François Ochsenbein pour le groupe de travail VOTable.
– André Schaaff pour le groupe de travail Grid and Web Services.

10. les Unified Column Descriptors permettent de décrire de façon unique les différentes colonnes des tables de données astronomiques (dans l'ensemble des tables de VizieR il y a en moyenne 60 descriptions différentes par colonne pour en réalité le même concept).

Chapitre 2

Déroulement du stage

2.1 Initialisation du stage

2.1.1 Sujet de stage

Le Centre de Données Astronomiques de Strasbourg a une forte activité de recherche et développement depuis de nombreuses années afin de suivre les évolutions techniques. Actuellement, il évalue, entre autres, l'utilisation de ses services en mobilité (smartphones, tablettes tactiles..), l'utilisation des écrans multitouch (sur terminaux mobiles mais également sur terminaux traditionnels) et avec une priorité moindre, les possibilités d'affichage offertes par les écrans 3D.

Dans ce cadre, le stage avait pour but d'évaluer l'utilisation du multitouch sur différents types de terminaux et plus particulièrement pour le logiciel Aladin, l'atlas interactif du ciel. Après l'acquisition d'un écran tactile, André Schaaff s'est rendu compte qu'il n'y avait pas besoin d'adapter l'interface d'Aladin aux écrans tactiles dans un premier temps. En effet, le système d'exploitation Windows Seven possède déjà une interface transformant les événements tactiles en événements équivalents de souris. Des tests menés avec la version standard d'Aladin ont mis en avant le fait qu'il n'était pas nécessaire d'apporter des modifications à l'outil. Des tests seront également menés avec les dernières versions de Linux car Aladin ne cible pas une plateforme particulière. Aladin n'ayant pas été pensé pour une utilisation tactile, des adaptations seraient utiles afin d'en faciliter l'utilisation (emplacements des composants graphiques : boutons, zones de saisies, etc..). Diverses pistes comme la création d'une version simplifiée adaptée au multitouch sont à l'étude pour une utilisation dans le cadre des planétariums, écoles, etc..

Le sujet du stage s'est donc tourné vers l'adaptation du logiciel Aladin vers

les terminaux mobiles (smartphones, tablettes Internet) possédant le système d'exploitation Android. Le but était d'évaluer la part du code java d'Aladin "adaptable" à cet environnement mais également d'évaluer les performances sur des plateformes relativement modestes comparées aux ordinateurs de bureaux et aux portables.

2.1.2 Moyens mis à disposition

Durant les premiers jours du stage, il a fallu comprendre les besoins des utilisateurs des outils développés par le CDS. C'est pourquoi, les autres stagiaires et moi même avons assisté à une présentation des différents services fournis par le CDS. Devant travailler sur le petit frère d'Aladin, il était primordial que je comprenne les fonctionnalités de celui-ci. J'ai donc été briefé par Pierre Fernique, le développeur principal d'Aladin. J'avais à disposition des codes sources et divers moyens de documentation.

Codes sources

Les codes sources à disposition étaient essentiels, ils m'ont permis d'utiliser des méthodes de classe toutes faites, voire même de m'inspirer d'algorithmes déjà mûrement réfléchis.

J'ai bien entendu récupéré le code source d'Aladin. J'avais également à disposition une version simplifiée qui permettait d'afficher le maillage d'une sphère en 3D, tout en ayant la possibilité de se déplacer autour et de zoomer avec la molette (cf figure 2.1).

Moyens de documentation

Je disposais du livre "Programmation Android" des éditions Eyrolles, qui m'a été très utile tout le premier mois. Ensuite, j'ai aussi beaucoup utilisé l'API java d'Android qui est décrite dans la documentation officielle en ligne (http ://developer.android.com), celle-ci est bien entendu en anglais. Elle est très fournie et simple d'utilisation.

Cependant, la technologie étant jeune, j'ai remarqué qu'il était assez long de trouver des réponses à ses problèmes. Il est rare de trouver un forum avec un problème proche de celui que l'on recherche, et il est encore plus rare que celui-ci soit résolu. Je faisais pourtant la grande majorité de mes recherches en anglais. A titre d'exemple, je trouve que les forums de développement sur iPhone sont plus actifs.

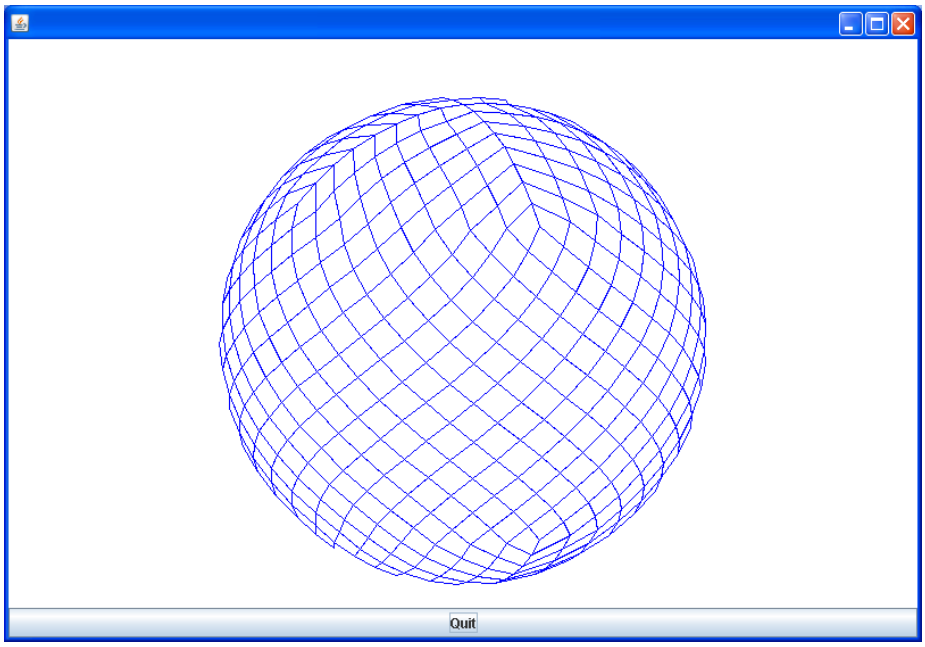

FIGURE 2.1 – Version simplifiée d'Aladin (mode Allsky)

Moyens matériels

Sans compter le bureau, l'ordinateur, le fauteuil confortable, et la vue sur le parc, on m'a fournis du matériel parfaitement adapté au sujet de mon stage. Même si mon stage ne portait plus tout à fait sur ce sujet, j'avais un écran multitouch Iiyama 22 pouces. Et j'avais surtout une tablette tactile Archos 101it équipée d'Android 2.2.1, celle-ci fut très utile.

2.1.3 Éléments techniques

Même si je n'ai pas compris dès le départ ces éléments techniques, je vous les décris ici, afin que vous compreniez mieux la suite.

Sphère Healpix

Les astronomes représentent le ciel étoilé comme un globe. Sur cette sphère, ils appliquent différentes projections (sinus, tangentielle, aitoff, cartésienne, ncp, zpn..), la plus utilisée étant la tangentielle.

Pour dessiner en 3D cette sphère et y appliquer des images, il faut découper

la sphère en carrés ou triangles. La sphère Healpix est une forme de découpage de sphère. HEALPix vient de Hierarchical Equal Area isoLatitude Pixelization of a sphere. Elle a été conçue en 1997 par Krzysztof M. Górski au Theoretical Astrophysics Center (TAC) à Copenhague (Danemark). La sphère Healpix a la particularité d'être composée uniquement de carrés, il est ainsi plus simple de la subdiviser [1] tout en gardant le contrôle de l'affichage des images (cf figure 2.2). D'autres types de sphère existent, mais ne sont pas forcément adaptées pour appliquer des images "régulières" (cf figure 2.3).

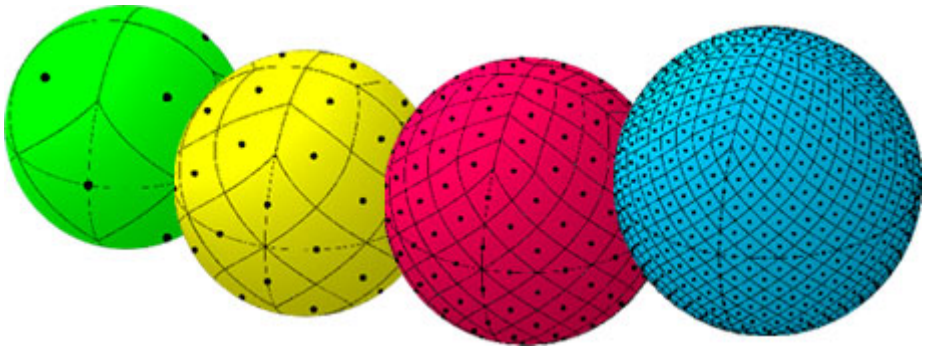

FIGURE 2.2 – Subdivision d'une sphère Healpix (réalisé par la NASA)

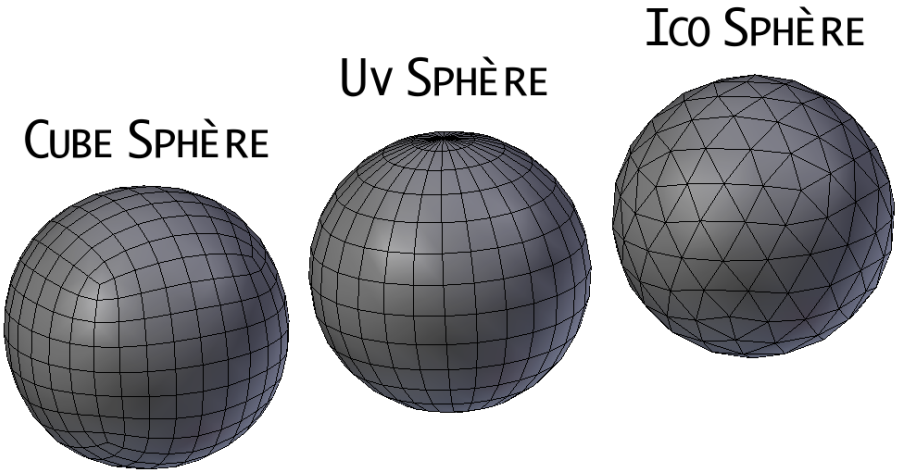

FIGURE 2.3 – Autres types de sphère polygonale

1. Une subdivision consiste à diviser chaque arête, si ce sont des triangles, le nombre de faces est triplé, si ce sont des carrés, le nombre de faces est quadruplé.

Système de coordonnées

En astronomie, pour décrire la position d'un objet, on utilise les coordonnées célestes (cf figure 2.4) : la déclinaison et l'ascension droite, ou encore la longitude et la latitude. Ça peut être assimilé à des coordonnées sphériques en géométrie. Lorsque j'utilisais ces angles, ils étaient exprimés en degrés. La longitude est comprise entre 0 et 360 degrés, tandis que la latitude est comprise entre -90 et 90 degrés.

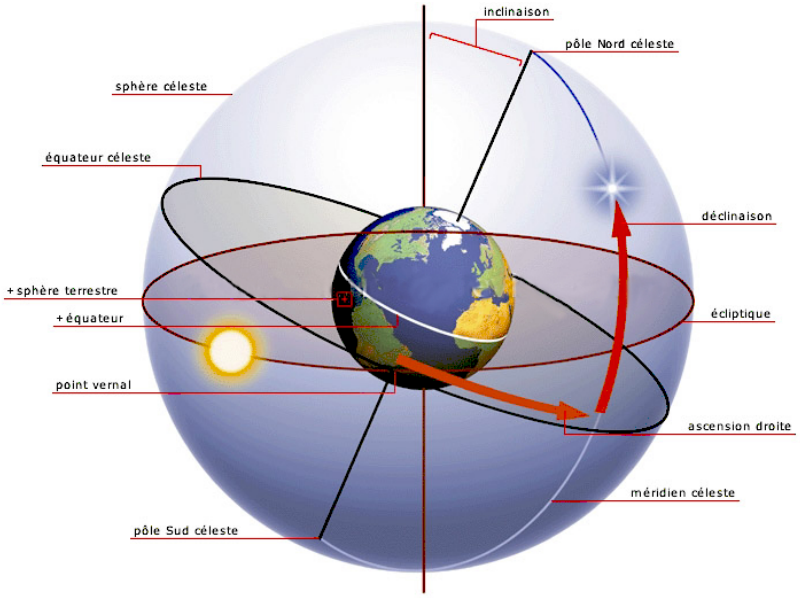

FIGURE 2.4 – Système de coordonnées célestes (www.ledictionnairevisuel.com)

2.2 Premiers prototypes

2.2.1 De Swing à Android

Une adaptation rapide

Là où la principale concurrence nécessite que l'on s'équipe de son système d'exploitation et apprenne un langage propre à la marque, Android a l'avantage d'être rapide à utiliser pour quelqu'un habitué à java et son environnement incontournable, Eclipse.

Il faut d'abord télécharger et installer le SDK d'Android (http ://developer.android.com/sdk).

Pour créer un premier projet, un assistant dans Eclipse génère les fichiers de base. Dans "AndroidManifest.xml", on configure la base de l'application. Dans le dossier res, on peut placer toutes les ressources qui nous sont utiles (images, styles d'interface, chaînes de caractère selon la langue..). La compilation génère automatiquement la classe R faisant le lien avec les fichiers des ressources. Il est ensuite assez aisé de retrouver les classes et méthodes équivalentes du package Swing. Pour le débugage, la classe Log permet d'afficher du texte sur la console.

Compilation et tests

Avec Eclipse, la compilation est presque identique à un projet normal. Il est aussi tout à fait possible d'utiliser la fonction de debug d'Eclipse. Pour lancer l'application, il faut simplement choisir "l'appareil" ciblé.

Si l'on ne dispose pas de smartphone ou de tablette sous Android, on peut créer et configurer une machine virtuelle. Celle-ci est extrêmement lente, et cela même pour virtualiser son propre système d'exploitation, Android. Avec le SDK, il existe pourtant un éditeur de machine virtuelle permettant de choisir exactement le matériel que l'on utilise, mais ça ne change rien. Il semblerait que ce problème soit moins marqué sur Linux. Et là, vous me permettrez de comparer une dernière fois avec la technologie d'Apple, car la machine virtuelle iPhone est quant à elle, vraiment réaliste.

Je n'ai pas eu longtemps à m'en soucier, j'avais à disposition la tablette Archos dès la seconde semaine. Pour pouvoir lancer directement après la compilation le programme sur la tablette, il suffit que la tablette soit connectée en USB et que l'ordinateur possède le driver ADB du constructeur.

Adaptation de la version simplifiée

Après quelques rapides essais de projets Android, j'ai adapté la version simplifiée d'Aladin. Ça consistait essentiellement à retrouver les équivalents des classes et méthodes du package Swing, comme par exemple la classe JFrame qui a pour équivalent Activity et JPanel qui a pour équivalent View. J'utilisais OnClick-Listener pour être à l'écoute des boutons, et OnTouchEvent pour écouter les événements de "touché". Afin de ne pas polluer le code java, la mise en page était décrite dans un fichier XML situé dans les ressources.

Premiers résultats

L'exécution était très lente alors que l'on n'affichait pas encore une seule texture, et que le niveau de subdivision de la sphère était bas. Une des raisons de cette lenteur pouvait venir du fait que les calculs de projection 3D se faisaient à partir

du processeur. Afin d'utiliser les atouts de la puce graphique, il fallait utiliser la librairie minimisée OpenGL-Es.

J'avais déjà développé avec OpenGL, alors j'ai proposé de continuer le développement avec cette technologie. Mes maîtres de stage auraient préféré que l'on ne passe pas par cette technologie, en effet elle impliquait de devoir revoir une bonne partie de l'aspect graphique d'Aladin. Mais comme j'avais de l'avance sur le travail à effectuer, ça ne les dérangeait pas que je fasse des tests la semaine suivante.

2.2.2 OpenGL-Es

L'API d'Android possède des classes simplifiant légèrement l'implantation d'OpenGL-Es dans son application. La classe GLSurfaceView.Renderer permet de décrire le rendu 3D. Elle est composée des méthodes onSurfaceCreated, où l'on configure OpenGL, onSurfaceChanged, où l'on configure généralement la taille de l'écran, et onDrawFrame, où l'on décrit le dessin 3D à réaliser à chaque frame [2]. La classe GLSurfaceView doit posséder un objet Renderer, cette classe est utilisée pour gérer les interactions avec l'écran. C'est d'ailleurs là où j'utiliserai onTouchEvent, la méthode à l'écoute des événements de touché. Ces classes simplifient l'implantation d'OpenGL-Es, car elles gèrent toutes seules le Thread [3] et la boucle d'affichage.

Voir en annexe (A.1), un exemple d'utilisation de base d'OpenGL-Es. Dans cet exemple, nous utilisons l'objet de la classe GL10 pour pouvoir faire des appels de fonction OpenGL, plus tard, nous n'utiliserons plus cette classe qui implique de devoir transmettre l'objet, nous utiliserons plutôt la classe statique GLES10.

Principes de base de la 3D polygonale

OpenGL, tout comme son concurrent Directx, utilisent les concepts de la représentation polygonale, pour rendre un effet de 3 dimensions. Cela consiste à représenter la surface d'un objet visuel quelconque en plusieurs polygones, eux-même composés de vertices. Un vertex (vertices, au pluriel) est un point situé dans l'espace à partir de 3 coordonnées dans une base orthonormée. Pour former des faces, on relie explicitement chaque vertex à ses voisins. Pour donner de la couleur à cette surface, il est possible de donner une couleur RGB [4] par vertex. Il est aussi possible d'appliquer une texture image sur l'objet, en donnant à chaque vertex, des coordonnées 2D sur l'image.

2. Une frame représente une image immobile, en faisant défiler les frames les unes derrières les autres on trompe la perception visuelle de l'œil humain.

3. Un Thread est un processus avec une tâche spécifique qui s'exécute en parallèle des tâches des autres threads.

4. Couleur décrite à l'aide de sa valeur rouge, verte et bleue.

Les Vertex Arrays

Dans les premières versions d'OpenGL, les fonctions de dessin étaient relativement simples à comprendre, mais avaient le désavantage de faire des appels de fonctions beaucoup trop nombreux. En effet, rien que pour informer des coordonnées spatiales des vertices, il fallait faire un appel de fonction par vertex.

```
1 glBegin(GL_TRIANGLES);
2 glVertex3f(1.0,1.0,0.0);
3 glVertex3f(0.0,1.0,0.0);
4 glVertex3f(0.0,0.0,0.0);
5 glEnd();
```

Alors qu'avec les Vertex Arrays, il suffit de pointer vers un tableau de données.

```
1 glVertexPointer(vertex_size, GL_FLOAT, 0, vertexBuffer);
```

Même si je ne l'ai pas utilisé, il existe une méthode encore plus évoluée, celle des Vertex Buffer Objects. Elle fonctionne comme les Vertex Arrays, sauf que l'on stocke les données dans la mémoire de la puce graphique, ce qui rend l'accès plus rapide. Cependant cette méthode n'est disponible qu'à partir d'OpenGL-Es 2.0, or certains smartphones ne sont compatibles qu'OpenGL-ES 1.0. Il se peut que je consacre du temps à l'exploitation de cette fonction dans le cadre du CDD, qui a m'a été proposé pour l'été.

2.2.3 Développement d'une application de comparaison

Afin de comparer les différences de performances entre l'utilisation ou non d'OpenGL-Es, j'ai développé une application adaptée (cf figure 2.5).

Une interface adaptée à la comparaison

L'interface est composée de boutons permettant de passer de la vue OpenGL à la vue normale. Elle comporte aussi un contrôleur servant à sélectionner le niveau de subdivision de la sphère Healpix.

Un Thread est créé afin d'afficher toutes les secondes la rapidité de rafraichissement du dessin. Cette rapidité est exprimée en FPS, c'est à dire en frame par seconde.

Application des textures de test

Etant donné qu'il est primordial qu'Alipad affiche des textures sur ses losanges, il était important de tester l'impact des textures sur le rafraichissement de l'image.

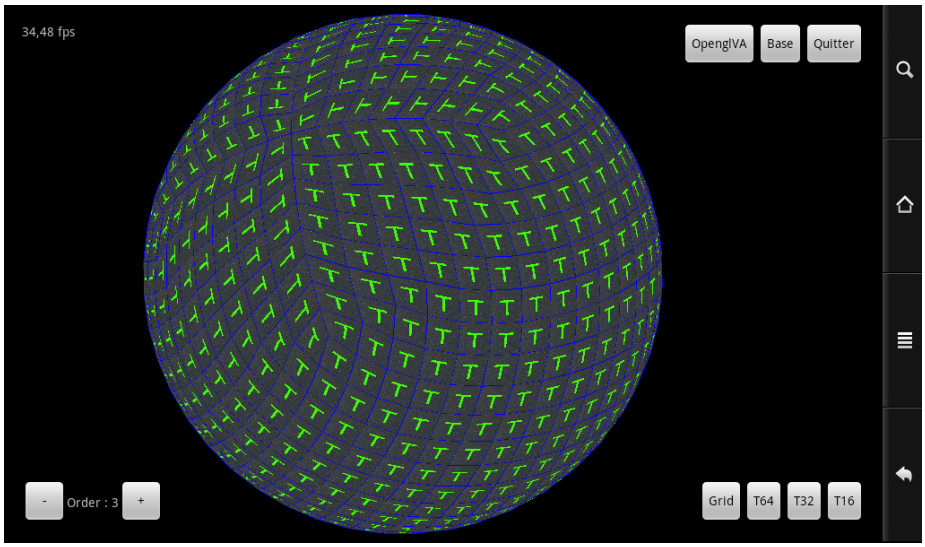

FIGURE 2.5 – Application de comparaison entre un affichage OpenGL-Es et un affichage normal

Et cela en prenant soin que chaque losange charge la texture de test comme si cette texture n'avait pas encore été chargée.

Résultats des comparaisons

Pour que l'affichage soit suffisamment fluide pour l'œil d'un humain, il faut que l'animation atteigne au moins les 25 frames par seconde. Ensuite pour que la sphère paraisse lisse, il faut que le premier niveau de subdivision soit de 3. A ce niveau de subdivision la sphère doit pouvoir accueillir des textures de 64 pixels de côté. Voici donc un tableau comparatif des résultats, avec un affichage du maillage et de la texture de 64 pixels de large.

Niveau de subdivision	Version normale (en FPS)	Version OpenGL-Es (en FPS)
1	9	350
2	2,2	125
3	0,8	30
4	0,2	9,5

On constate que le gain de performances est vraiment intéressant. Les attentes concernant l'affichage global de la sphère au niveau de subdivision 3 sont remplies. Il est maintenant admis que la suite du développement s'effectuera avec OpenGL-Es.

2.3 Développement d'Alipad

2.3.1 Première version

En se basant sur la partie OpenGL de l'application de comparaison, j'ai monté une première version permettant simplement d'afficher le niveau de subdivision 3 de la sphère avec les premier niveau de textures. Cette version possédait un système de zoom simple (le système de zoom final est expliqué plus loin).

Cette version a été présentée à la session éducation de la conférence IVOA le 19 mai à Naples. Nous avions aussi préparé des vidéos de démonstration d'Aladin avec un écran tactile. Les vidéos sont disponibles à l'adresse suivante : www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/InterOpMay2011VOandEducation

2.3.2 Zoom et restriction de maillage

OpenGL-Es a beau offrir un affichage correct à un certain niveau de subdivision de la sphère, ça ne suffirait pas dans le cas du zoom, où la sphère doit se subdiviser proportionnellement au facteur de zoom. Or si l'on dépasse le niveau de subdivision 3, le nombre de polygones à afficher augmente significativement tout comme le temps de rendu. Il faut donc restreindre les polygones affichables à ce qui est visible par la caméra (cf figure 2.6).

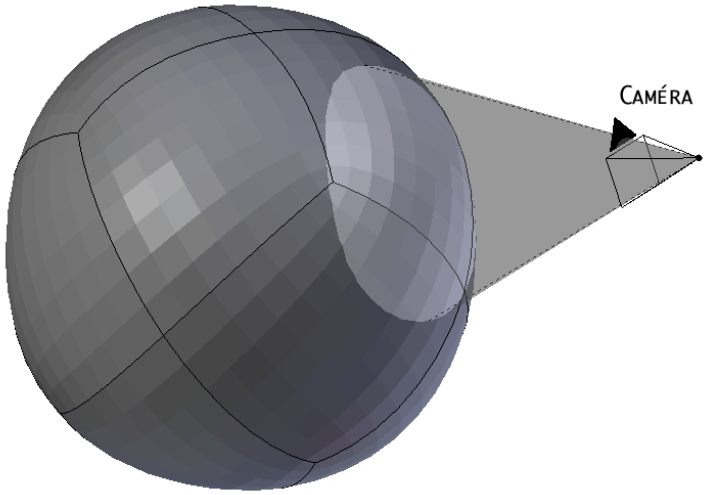

Figure 2.6 – Schéma décrivant la restriction de maillage

Système de zoom

Le système de zoom est géré principalement dans la fameuse méthode captant les événements de "touché", onTouchEvent(MotionEvent e). Pour connaitre le nombre de doigts touchant l'écran, on utilise la méthode getPointerCount() de l'objet MotionEvent. Lorsque l'on détecte 2 doigts, on active un booléen dédié au zoom, et on calcule la distance entre ces deux doigts. Ensuite au moment de détecter un mouvement des doigts, on fait la différence entre l'ancienne et la nouvelle distance les séparant, on obtient ainsi le facteur de zoom.

Une fois le facteur de zoom déterminé, on demande au rendu d'en tenir compte. Celui-ci multiplie d'abord l'ancien facteur de zoom par le nouveau, afin d'obtenir une vitesse de zoom proportionnelle au zoom courant. Ensuite il détermine si ce nouveau facteur de zoom nécessite que l'on modifie le niveau de subdivision. Enfin, au moment de rendre la scène, je reporte le facteur de zoom dans la fonction OpenGL servant à configurer la projection 3D :

```
1 glOrthof(-1/zoom, 1/zoom, -1/zoom, 1/zoom, -1f, -1+1/((zoom > 200)
      ? 200 : zoom));
```

Les 4 premiers paramètres indiquent les positions des bords de la caméra. Les 2 derniers définissent la distance minimale et maximale de vision.

Placement de la caméra

En vue d'une restriction du maillage dépendant du zoom et du placement de la caméra, il était important de revoir complètement le déplacement de cette dernière. La caméra devait être définie par deux vecteurs : un vecteur décrivant ce qu'elle pointe, et la coordonnée céleste de ce qu'elle voit, c'est à dire le vecteur opposé du premier vecteur. A chaque rotation de la caméra il fallait donc calculer ces vecteurs, et les rendre compatibles avec le système de coordonnées célestes, c'est à dire compris entre 0 et 360 degrés pour la longitude et -90 et 90 degrés pour la latitude. Pour que le rendu tienne compte de la position de la caméra, il fallait utiliser une fonction de la classe GLU [5] :

```
1 GLU.gluLookAt(gl, 0, 0, 0, cam.x, cam.y, cam.z, 0, 0, cam_z_signe);
```

Les trois premiers 0 indiquent que la caméra pointe vers l'origine. Les trois paramètres suivants concernent la position de la caméra. Les trois derniers définissent la rotation de la caméra sur elle-même.

5. GLU (OpenGL Utility Library) est une bibliothèque associée à OpenGL. Elle vient compléter cette dernière en apportant quelques routines pour des opérations de plus haut niveau.

Restriction du maillage

Pour bien comprendre le changement fait à ce niveau, il faut savoir qu'avant, les coordonnées de points de la sphère étaient déterminées globalement, elles étaient ensuite enregistrées et utilisées régulièrement par OpenGL. Il était maintenant question de séparer en deux classes les données géométriques de la sphère. Une classe dédiée aux losanges, nommée Pix, avec dedans le niveau de subdivision et l'identifiant. Les coordonnées spatiales des 4 coins du losanges ne sont pas retenues, afin de ne pas saturer la mémoire vive. Une autre classe, nommée Sky, avec dedans une liste des losanges à afficher, et les buffers utiles au dessin OpenGL. Pour déterminer la structure de ces données, cela n'a pas été simple, car il fallait garder la structure des buffers utilisables avec les méthodes "arrays" d'OpenGL.

Une des complexités de ce système réside aussi dans le fait que pendant qu'un Thread met à jour les données à la suite d'un déplacement de la caméra, le Thread d'affichage OpenGL peut souvent être en train de rafraichir le rendu. Dans un premier temps j'avais mis un booléen activé le temps de remplacer les anciens buffers par les nouveaux, mais il fallait vérifier trop souvent l'état de ce booléen dans des opérations qui demandaient de la rapidité. Je me suis donc tourné vers une autre solution : un booléen indiquant si les buffers sont actuellement utilisés par le Thread OpenGL, le remplacement des buffers se fait une fois que ce booléen est à faux.

Pour sélectionner les losanges à afficher, j'utilisais une fonction d'Aladin, celle-ci prenait en paramètres le centre de la zone en coordonnées célestes, l'angle du cône à sélectionner, et le niveau de subdivision de la sphère. Cette fonction me retournait un tableau rempli des identifiants des losanges concernés.

2.3.3 Gestion des textures

L'intérêt d'avoir un maillage de losanges est de pouvoir appliquer une texture sur chaque losange. D'ailleurs l'affichage du maillage existe uniquement dans le cadre du débogage. On peut donc dire que la partie précédente dédiée à la gestion du maillage fait aussi parti du système de gestion de textures. A terme l'application doit juste être capable d'afficher un relevé photographique du ciel, avec possibilité d'avoir des renseignements sur un objet céleste sélectionné.

OpenGL et les textures

Avec OpenGL, on applique les textures sur des faces. Chaque vertex est composé de coordonnées 2d : généralement appelées coordonnées "u, v". Les coordonnées uv décrivent la position de la vertex sur une image donnée (cf figure 2.7).

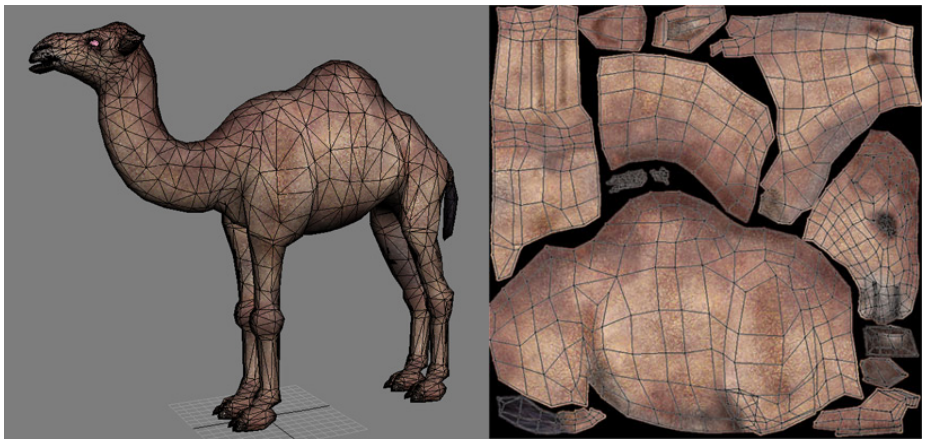

FIGURE 2.7 – Exemple d'application de texture en 3d (réalisé par Nicky Pearson)

Dans mon cas, les textures sont générées avec Aladin. Il existe d'abord une texture globale au niveau de subdivision 3 qui regroupe toutes les textures de 64 pixels de côté dans une seule image, pour simplifier j'ai surnommé ce niveau de subdivision, le niveau 2.5. Ensuite il existe un jeu d'images de 512 pixels de côté pour les niveaux suivants, si le premier niveau est surnommé 2.5, c'est parce qu'il existe aussi un jeu d'images précises pour le niveau de subdivision 3.

Pour pouvoir utiliser des images avec OpenGL, il faut préparer un espace dédié à ces images. Sachant qu'au niveau 2.5, on affiche la sphère complète, soit 768 losanges, et qu'aux niveaux suivants on affiche, en moyenne 250 losanges. Il faut donc créer au maximum 768 pointeurs vers des textures. La fonction suivante est utilisée dans ce cadre là :

```
1 texture_index_table = new int[768];
2 GLES10.glGenTextures(768, texture_index_table, 0);
```

Pour remplir une texture, à partir d'une image de type bitmap (bmp, png, et jpg en filtrant certaines données) on utilise le code suivant :

```
1  public void loadTexture(Bitmap bmp, int index_in_table) {
2      // Selection de la texture
3      GLES10.glBindTexture(GLES10.GL_TEXTURE_2D,  index_in_table);
4
5      // Creation des mipmaps
6      GLES10.glTexParameterf(GLES10.GL_TEXTURE_2D, GLES10.
           GL_TEXTURE_MIN_FILTER, GLES10.GL_LINEAR);
7      GLES10.glTexParameterf(GLES10.GL_TEXTURE_2D, GLES10.
           GL_TEXTURE_MAG_FILTER, GLES10.GL_LINEAR);
8
9      // Parametrage application des textures
10     GLES10.glTexParameterx(GLES10.GL_TEXTURE_2D, GLES10.
           GL_TEXTURE_WRAP_S, GLES10.GL_CLAMP_TO_EDGE);
```

```
11      GLES10.glTexParameterx(GLES10.GL_TEXTURE_2D, GLES10.
            GL_TEXTURE_WRAP_T, GLES10.GL_CLAMP_TO_EDGE);
12      GLES10.glTexEnvf(GLES10.GL_TEXTURE_ENV, GLES10.
            GL_TEXTURE_ENV_MODE, GLES10.GL_REPLACE);
13
14      GLUtils.texImage2D(GLES10.GL_TEXTURE_2D, 0, bmp, 0);
15      bmp.recycle();
16  }
```

Les textures mipmaps remplacent la texture originale lorsque celle-ci est trop loin de la caméra pour être affichée correctement (cf figure 2.8). L'option GL_LINEAR précise que pour générer ces mipmaps, OpenGL fait la moyenne linéaire pondérée des 4 pixels les plus proche du centre du pixel. Cette technique peut aussi donner un effet de profondeur.

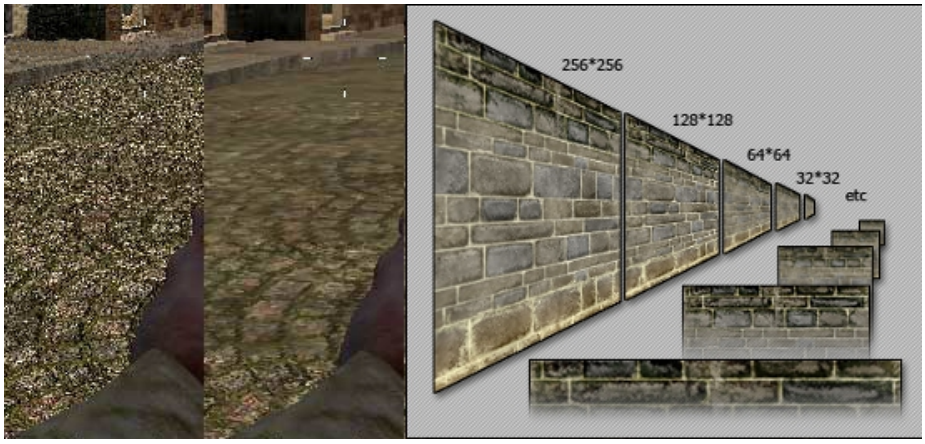

FIGURE 2.8 – Effet mipmap (à gauche tom's hardware guide, à droite Josh Short)

Enfin, pour pouvoir utiliser les textures précédemment chargées lors du dessin, il fallait d'abord envoyer les coordonnées uv dans l'ordre des vertices. Ensuite il fallait pointer vers la texture concernée et enfin afficher la face qui va avec.

```
1  GLES10.glTexCoordPointer(2, GLES10.GL_FLOAT, 0, uv_texture_buffer);
2  for (int i=0 ; i<npix ; i++) {
3      if ((pix_drawable.get(i).texture.state == Texture.S_DRAWABLE) {
4          GLES10.glBindTexture(GLES10.GL_TEXTURE_2D, pix_drawable.get(i
               ).texture.textid);
5          face_index_buffer.position(i*4);
6          GLES10.glDrawElements(GLES10.GL_TRIANGLE_STRIP, 4, GLES10.
               GL_UNSIGNED_SHORT, face_index_buffer);
7      }
8  }
```

Lorsque je faisais des appels de fonction OpenGL en rapport avec les textures, souvent je n'étais pas dans le Thread OpenGL d'origine, ce qui pouvait causer

un crash inexpliqué. J'ai donc appris qu'il était possible d'envoyer des tâches au Thread OpenGL avec la fonction suivante :

```
gl_context.queueEvent(new Runnable() {
    public void run() {
        // taches OpenGL
    }
}
```

Téléchargement et mise en cache des textures

Pour pouvoir gérer l'affichage des textures, les tâches étaient réparties sur trois Threads : Un premier affichant quelque soit le niveau de subdivision la portion de la texture globale du niveau 2.5, l'affichage est quasiment instantané, ce qui permet d'éviter d'avoir trop longtemps des faces noires. Un second chargeant la texture depuis l'image en cache si elle existe au niveau demandé ou à un niveau moins bien défini. Et un troisième téléchargeant l'image depuis internet si elle n'existe pas en cache et si une connexion à internet est possible (cf figure 2.9).

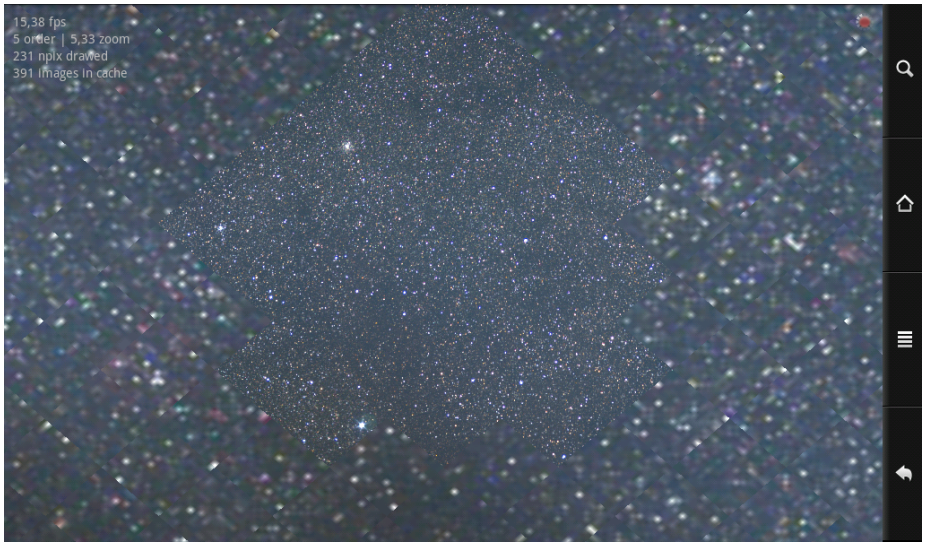

FIGURE 2.9 – Exemple de chargement d'une zone

Ces Threads causent certains plantages. Actuellement chaque Thread possède une liste chaînée contenant les textures qu'il doit traiter, une fois traité il transmet la texture au Thread concerné. Ce système implique de nombreuses mises à jour sur les listes chaînées et parfois par deux Threads étrangers, il faut régulièrement trier la liste pour mettre en premier les textures les plus proches de la position de la caméra. Dans Aladin, j'ai appris qu'un seul thread possédait

cette liste de textures. Durant mon CDD, je prévois donc d'adopter un système similaire, ce qui devrait réduire le nombre d'erreurs.

La mise en cache des fichiers est indispensable. Ce genre d'application doit pouvoir être utilisable à l'extérieur sans devoir de se connecter à Internet via un réseau. D'ailleurs au quotidien, l'ensemble des utilisateurs d'Aladin visualisent environ 10 000 textures, cela représente plusieurs centaines de Mo [6] de données, mais seulement quelques dizaines de Mo sont téléchargés le reste étant repris du cache local. Les astronomes ont donc l'habitude de revenir sur des données déjà étudiées.

J'ai aussi ajouté un indicateur de connexion qui se met en vert lorsque la connexion à internet est faisable et en rouge lorsque ce n'est pas possible. Avec celui-ci, il y a aussi un indicateur de chargement, actif lorsque les 3 Threads de chargement de texture travaillent. Dans les paramètres, il est aussi possible de choisir le nombre maximal de textures à mémoriser en cache. Les textures les plus anciennement téléchargées sont alors supprimées. Il serait d'ailleurs intéressant d'ajouter un système datant la dernière utilisation d'une texture en cache, afin de supprimer plutôt les textures les plus anciennement utilisées.

L'interface est agrémentée d'un panneau de sélection permettant de choisir parmi plusieurs relevés photographiques. Et enfin, même si ça ne concerne pas la gestion des textures, j'ai ajouté une barre de recherche permettant pour l'instant uniquement de déplacer la caméra à une coordonnée céleste voulue.

6. Mégaoctet (Mo) est un multiple de l'unité mesurant la quantité de données en informatique, l'octet.

Conclusion et perspectives

L'objet du stage était de déterminer comment adapter le logiciel Aladin au système d'exploitation Android. Afin de répondre à la problématique, j'ai donc d'abord tenté d'adapter le code du logiciel Aladin à l'API d'Android, mais les calculs de projection 3D par le processeur demandèrent trop de performances pour du matériel portable. Une autre solution consistait à utiliser la librairie graphique OpenGL, celle-ci utilise la puce graphique pour les calculs de masse. Ainsi pour déterminer quelle pouvait être la solution la plus adaptée, j'ai développé une application permettant de comparer les performances entre le système d'affichage d'origine et l'affichage avec OpenGL.

La fluidité d'affichage souhaitée fut obtenue avec la version OpenGL, l'application allait donc être développée dans ce sens. L'inconvénient de l'utilisation d'OpenGL était qu'il a fallu revoir la majeure partie des algorithmes liés à l'affichage 3D. L'un des plus importants concernait la restriction des parties du globe à afficher selon le placement et le zoom de la caméra. Il a fallu aussi adapter la gestion du chargement des images à appliquer sur la sphère.

Une version distribuable étant maintenant envisageable, mes maîtres de stage m'ont ainsi proposé un CDD pour les vacances d'été afin d'améliorer la version actuelle de l'application. Je compte notamment ajouter la possibilité d'obtenir des informations sur un objet céleste sélectionné, améliorer la barre de recherche en conséquence et corriger des bugs liés aux Threads de chargement de textures. Il est aussi prévu que l'application tienne compte des différents capteurs intégrés à la tablette, ainsi on pourrait imaginer l'application réagissant à la position et l'inclinaison de l'écran. D'autre part, le CDS envisage de développer une version HTML5, qui serait ainsi compatible pour tout type de smartphone.

Mon stage s'est très bien déroulé. J'ai évolué dans un domaine qui me passionnait et où mes expériences passées m'ont été utiles. J'avais la chance d'être dans une équipe disponible et à l'écoute, où l'on ne se sentait jamais abandonné.

Ce stage m'a aussi permis de découvrir le monde de la recherche au sein d'un laboratoire. J'ai aussi assimilé l'utilité des outils informatiques pour des domaines scientifiques tel que l'astronomie.

Références bibliographiques

Programmation Android - De la conception au déploiement avec le SDK Google Android 2 des Editions Eyrolles, par Damien Guignard, Julien Chable, Emmanuel Roblès, Nicolas Sorel, Vanessa Conchodon.

developer.android.com Documentation officielle d'Android ; contient notamment l'API Java d'Android et des tutoriaux.

khronos.org/opengles Api d'origine d'OpenGL-Es.

stackoverflow.com Forum anglophone dédié à la programmation, très actif.

androidsnippets.com Forum anglophone dédié au développement sur Android.

forum.frandroid.com Forum francophone dédié à Android, avec une partie consacrée au développement d'applications.

Glossaire

Aladin Application développée en java, atlas 3D, interactif du ciel étoilé.

Alipad Application Android développée en java, se basant sur Aladin. C'est l'objet du stage.

Android (prononcé androïde) est un système d'exploitation open source pour smartphones, PDA et tablettes, conçu par Android, une startup rachetée par Google (définition de Wikipedia).

Cache Un fichier est en cache lorsqu'il est gardé un certain temps dans la mémoire de masse de l'appareil (dans un disque dur par exemple).

CDS Centre de Données astronomiques de Strasbourg ; collecte, homogénéise, distribue et préserve l'information astronomique pour le bénéfice de l'ensemble de la communauté astronomique.

Code source Ensemble d'instructions écrites dans un langage de programmation informatique de haut niveau, compréhensible par un être humain entraîné, permettant d'obtenir un programme pour un ordinateur (définition de Wikipedia).

Compilation Travail réalisé par un compilateur qui consiste à transformer un code source lisible par un humain en un fichier binaire exécutable par une machine (définition de Wikipedia).

Coordonnées célestes Composé de deux angles : la longitude (comprise entre 0 et 360 degrés) et la latitude (comprise entre -90 et 90 degrés).

Eclipse Environnement de développement principalement utilisé pour développé en Java.

Frame Une frame représente une image immobile ; en faisant défiler les frames les unes derrières les autres on trompe la perception visuelle de l'œil humain.

GLU OpenGL Utility Library ; Bibliothèque associée à OpenGL. Elle vient compléter cette dernière en apportant quelques routines pour des opérations de plus haut niveau.

GPL General Public License ; Licence informatique, libre mais oblige celui qui publie un logiciel utilisant ce code source à être également sous licence GPL.

Maillage (ou Mesh) est un ensemble de vertices, de faces, et d'arêtes.

Open source Désigne un programme libre de redistribution, avec accès au code source et aux travaux dérivés.

OpenGL Open Graphics Library ; c'est une spécification qui définit une API multiplate-forme pour la conception d'applications générant des images 3D voir 2D (définition de Wikipedia).

OpenGL-Es Open Graphics Library for Embedded System, parfois abrégé en OGLES ou GLES ; c'est une spécification du Khronos Group qui définit une API multiplate-forme pour la conception d'applications générant des images 3D dérivée de la spécification OpenGL, sous une forme adaptée aux plateformes mobiles ou embarquées (définition de Wikipedia).

Sphère Healpix Une sphère dont le maillage est formé à partir de subdivisions d'un volume composé de 12 losanges.

Standalone Une version Standalone est une application à part entière capable de fonctionner indépendamment de toute autre application.

Swing Package java possédant des classes dédiées à l'interface graphique.

Texture C'est un ensemble de pixels que l'on associe à une face. Les pixels sont bien souvent issus d'une image.

Thread Un Thread est un processus avec une tâche spécifique qui s'exécute en parallèle des tâches des autres threads.

Vertex (vertices au pluriel) Un point situé dans un espace à 3 dimensions.

Annexes

A.1 Exemple d'utilisation de base d'Opengl-Es

```
1  public class GLView extends GLSurfaceView {
2     private GLRenderer renderer;
3
4     public GLViewSky(Context context, AttributeSet attrs) {
5        super(context, attrs);
6        renderer = new GLRenderer(context, this);
7        setRenderer(renderer);
8     }
9
10    public boolean onTouchEvent(MotionEvent e) {
11       if (e.getAction() == MotionEvent.ACTION_DOWN)
12          // Le premier doigt vient de toucher l'ecran
13       else if (e.getAction() == MotionEvent.ACTION_UP)
14          // Le premier doigt vient de se retirer de l'ecran
15       else if (e.getAction() == MotionEvent.ACTION_MOVE)
16          // Le premier doigt est toujours sur l'ecran
17    }
18 }
19
20 public class GLRenderer implements Renderer {
21    private Objet3d objet3d;
22
23    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
24       objet3d = new Objet3d();
25       // Applique une couleur de fond noir
26       gl.glClearColor(0.0f, 0.0f, 0.0f, 1);
27       // Permet de ne pas afficher les elements caches par d'autres
28       gl.glEnable(GL10.GL_DEPTH_TEST);
29       // Active l'utilisation des vertex arrays
30       gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
31    }
32
33    public void onSurfaceChanged(GL10 gl, int w, int h) {
34       // Configure les dimensions et position de l'ecran de rendu
35       gl.glViewport(0, 0, w, h);
36    }
37
38    public void onDrawFrame(GL10 gl) {
```

```
39        // Efface l'ecran et le buffer de profondeur
40        gl.glClear(GLES10.GL_COLOR_BUFFER_BIT | GLES10.
              GL_DEPTH_BUFFER_BIT);
41        objet3d.draw(gl);
42    }
43 }
44
45 public class Objet3d {
46    private FloatBuffer vertexBuffer;
47    private ShortBuffer indexBuffer;
48
49    private float[] vertex = {-0.5f, -0.25f, 0, 0.5f, -0.25f, 0, 0,
          0.5f, 0};
50    private short[] index = {0, 1, 2};
51    private int nb_sommets = 3;
52
53    private int taile_float = 4;
54    private int taille_short = 1;
55
56    public Objet3d() {
57        // Initialise les buffers
58
59        ByteBuffer vbb = ByteBuffer.allocateDirect(nb_sommets *
              taille_float * 3);
60        vbb.order(ByteOrder.nativeOrder());
61        vertexBuffer = vbb.asFloatBuffer();
62        vertexBuffer.put(vertex);
63        vertexBuffer.position(0);
64
65        ByteBuffer ibb = ByteBuffer.allocateDirect(nb_sommets *
              taille_short * 2);
66        ibb.order(ByteOrder.nativeOrder());
67        indexBuffer = ibb.asShortBuffer();
68        indexBuffer.put(index);
69        indexBuffer.position(0);
70    }
71
72    public void draw(GL10 gl) {
73        // Envoie le buffer de points
74        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
75        gl.glColor4f(0, 0, 1, 1);
76        // Dessine l'image en indiquant les points que l'on relie
              pour former des faces
77        gl.glDrawElements(GL10.GL_TRIANGLES, nb_sommets, GL10.
              GL_UNSIGNED_SHORT, indexBuffer);
78    }
79 }
```

A.2 Captures d'écran d'Alipad