2012

TER – Création d'une librairie de Réalité augmentée

Pour application Android mobile

Aouizerate Erik & Faucher Julien

25/06/2012

Sommaire

1. Cadre de notre travail

Ce TER s'inscrit dans le cadre de notre première année de master WIC, à l'université Pierre Mendes France. Il se décompose en trois étapes réparties sur l'année scolaire. Ce rapport concerne la dernière partie et a pour but de finaliser notre travail. Il est proposé à l'initiative de Betül Aydin étudiante en thèse, au laboratoire d'informatique de Grenoble (LIG). Elle travaille dans l'équipe STEAMER avec Gensel Jérôme, professeur des universités en informatique et encadrant de sa thèse. Elle collabore aussi avec Tellez Bruno (Co-encadrant), maître de conférences, membre du laboratoire d'informatique en images et système d'information (LIRIS) à Lyon 2.

Ce projet porte sur la création d'une librairie de réalité augmentée pour les dispositifs mobiles sous Android, pour des applications géo-localisées. Cette librairie s'inscrit plus généralement dans le projet d'ARCAMA-3D (Augmented Reality Context Aware Mobile Application with 3D) « qui propose une approche intégrant la Réalité Augmentée (RA) comme dispositif principal associé à la présentation d'objets 3D »[1]. Ce projet « consiste en l'utilisation d'un modèle 3D comme un outil de RA qui facilite la compréhension de l'environnement et assure une interaction avec l'utilisateur »[1].

Plus précisément, l'application ARCAMA-3D permettra de géo-localiser et visualiser des objets virtuels de grande taille, tels que des bâtiments, au travers de la camera d'un Smartphone. A partir de ces objets, il sera possible de consulter des informations multimédia déposées par l'utilisateur lui-même ou un tiers. Ces informations seront accessibles sous forme de tag virtuel disposé en cloud lors de la sélection d'un objet. (cf. betul etc)

Pour une meilleure compréhension de la tâche qui nous a été impartie, cette entreprise peut être segmentée en trois grands domaines de réflexion qui sont représentés ci-dessous.

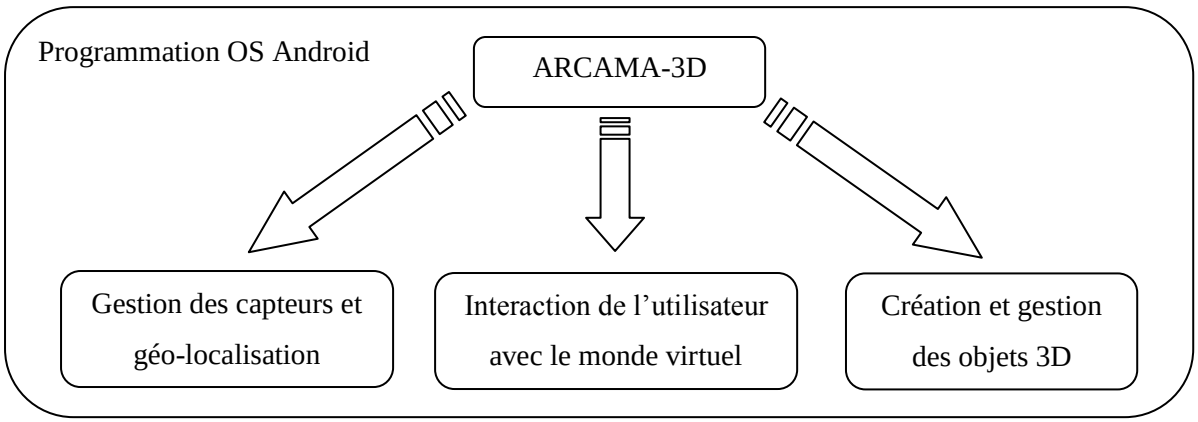

Figure 1 : diagramme des domaines de réflexion d'ARCAMA-3D

[1] Cf. Betül Aydin, 2012

Pour réaliser cette librairie il nous a fallu participer au domaine de réflexion sur la « gestion des capteurs et géo-localisation » et « interaction de l'utilisateur avec le monde virtuel », mais aussi appréhender la programmation pour terminaux mobile sous Android.

Au niveau des capteurs, notre travail s'est arrêté à l'analyse de ceux-ci et comment les faire fonctionner conjointement. Puis au niveau du deuxième domaine, le but a été de créer un objet 3D et de pouvoir interagir avec lui par le biais d'une sélection, et de ce fait implémenter quelques fonctions comme le changement de couleur de l'objet par exemple. Puis finalement d'intégrer toutes ces connaissances ensemble pour créer la librairie.

Le troisième domaine, le plus technique, concerne la création d'objets 3D complexes, leurs compressions, ainsi que l'amélioration de ces objets au point de vue esthétique (gain de réalisme en fonction de la lumière ou de l'ombre etc.). Nous n'avons pas pris part à cette partie là du projet car celle-ci sortait amplement de notre domaine de compétence. Contribuer à cette part de travail n'était pas pertinent car elle ne nous aurait amené aucune information sur le fonctionnement globale de l'application contrairement aux deux premiers domaines qui ont été primordiaux.

Pourquoi avons-nous choisi ce TER ? Quelles sont nos motivations ?

- Nous souhaitions travailler sur un domaine en lien avec les dernières technologies et apprendre de nouveaux langages. Android nous a semblé être parfaitement adapté à cette problématique.
- Face à l'émergence des nouvelles pratiques liées aux Smartphones, la réalité augmentée semble avoir de nombreuses applications et offrir une expérience utilisateur supplémentaire.

Les missions qui nous ont été confiées se rapprochent du génie logiciel pour la création de la librairie. Les objectifs de ce stage étaient :

- Mettre en application les théories du génie logiciel et de la gestion de projet
- Appréhender de nouvelles connaissances hors du cadre scolaire
- Expérimenter le travail collaboratif dans un projet se rapprochant du domaine professionnel

2. Etudes des capteurs

Nous avons étudié 2 capteurs nécessaires à l'élaboration de notre librairie.

Le GPS : L'acronyme GPS signifie *Global Positioning System*, c'est un système de navigation qui utilise un réseau de satellites qui permet de géo-localiser l'appareil qui l'utilise. Il renvoie une longitude, une latitude et une altitude. La latitude est comprise entre 0 et 90 dégrées et correspond à un plan horizontal situé sur l'axe nord-sud de la planète. Un degré de latitude équivaut à environ 111 kilomètres. La longitude est comprise entre 0 et 180 degrés et indique un plan vertical sur l'axe est-ouest. La correspondance en kilomètres dépend de la latitude. A l'équateur, 1 degré vaut 111 kilomètres et cette distance décroit jusqu'à 0 lorsqu'on se dirige vers les pôles. L'altitude est exprimée en mètres et correspond à la distance verticale nous séparant du niveau de l'océan. Cette dernière mesure est très peu précise.

La boussole - gyroscope : La boussole et le gyroscope sont couplés au sein du même capteur. Celui-ci mesure la position angulaire de l'appareil dans l'espace par rapport au nord magnétique. Ce capteur peut avoir de 1 à 3 trois degrés de liberté et renvoie 3 valeurs.

- ✓ l'azimut : donne l'angle avec le Nord magnétique.
- ✓ le pitch : donne l'angle autour de l'axe des x. En français cela se dit le tangage.
- ✓ le roll : donne l'angle autour de l'axe des y. En français cela se dit le roulis.

Voici les plages de valeurs pour chaque degré de liberté :

- ✓ L'azimut varie entre 0 et 360°, il représente l'angle avec le nord dans le sens des aiguilles d'une montre.
- ✓ Le pitch varie entre -180 et 180°, il représente l'inclinaison haut-bas de l'appareil selon l'axe Y. On obtient les valeurs suivantes :
  - o la valeur 0, l'appareil est parallèle au sol, face vers le ciel
  - o la valeur +/- 180, l'appareil est parallèle au sol, face vers le sol
  - o la valeur 90, l'appareil est perpendiculaire au sol, tête vers le bas
  - o la valeur –90, l'appareil est perpendiculaire au sol, tête vers le haut
- ✓ Le roll varie entre -90 et 90, il représente l'inclinaison droite-gauche de l'appareil selon l'axe des X. On obtient les valeurs suivantes (quand sa face est vers le haut) :
  - o la valeur 0, l'appareil ne penche pas
  - o la valeur 90, l'appareil est penché à gauche
  - o la valeur -90 l'appareil est penché à droite.

3. Interaction avec le monde virtuel

Nous devons implémenter la création de deux types d'objets OpenGL. Il y aura des objets complexes appelés « **édifices** » qui représenteront des immeubles ou des monuments (figure 2). Le deuxième type d'objet sera des « **balises** » ou « **tags** » qui seront en lien avec un édifice (figure 3). Ces balises devront être affichées lors d'un clic sur un édifice. De cette manière, l'utilisateur pourra avoir des informations complémentaires de différents types au sujet du monument. Ces balises seront des liens vers des photos, des articles ou tout autre contenu situé sur Internet ou dans la mémoire du téléphone.

Figure 2 : Modèle 3D du Temple d'Auguste et Livie à Vienne(38)

Figure 3 : Exemple d'icône 3D de réseaux sociaux

Ces spécifications n'altèrent pas le cadre du travail précédemment effectué. Cependant elles modifient la structure de la librairie.

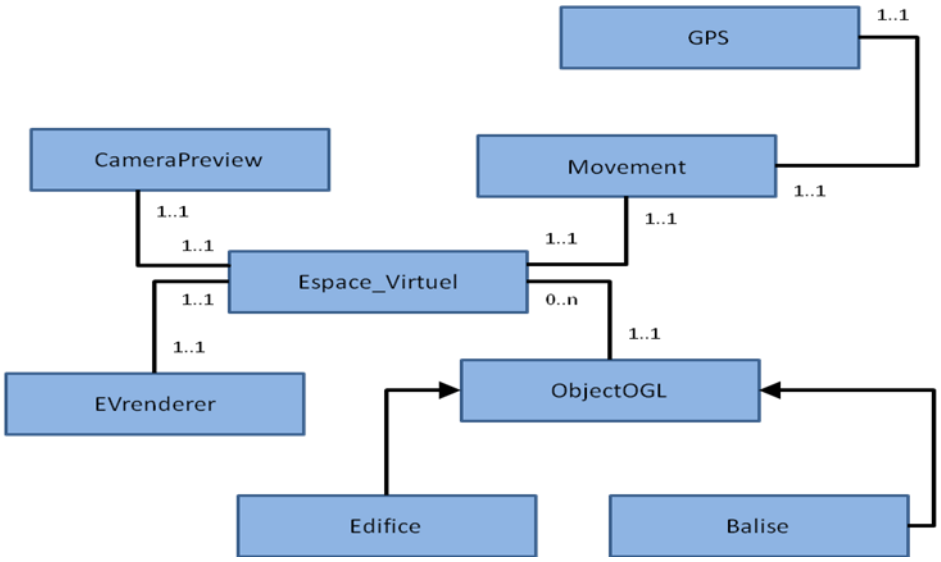

Figure 4 : Structure de la librairie

Voici le rôle de chaque classe de la librairie :

Espace_Virtuel est l'interface de la librairie. C'est la seul classe ayant des méthodes publiques. Elle a un rôle central et coordonne les interactions entre les différents objets.

CameraPreview permet de renvoyer sous forme d'une surface view le rendu de la camera.

EVrenderer permet de renvoyer sous forme d'une surface view les objets virtuels crées à l'aide d'OpenGL.

Movement permet de connaître les valeurs retournées par les capteurs et d'interagir avec le monde virtuel.

ObjectOGL est la classe mère des objets OpenGL. Cette implémentation permet de rajouter différents types d'objets et de tous les afficher dans EVrenderer.

Nous pouvons distinguer les interactions automatiques liées aux capteurs et les interactions ponctuelles (ou cas d'utilisations) dues à l'utilisateur. Les deux schémas suivant donnent un aperçu de ces interactions.

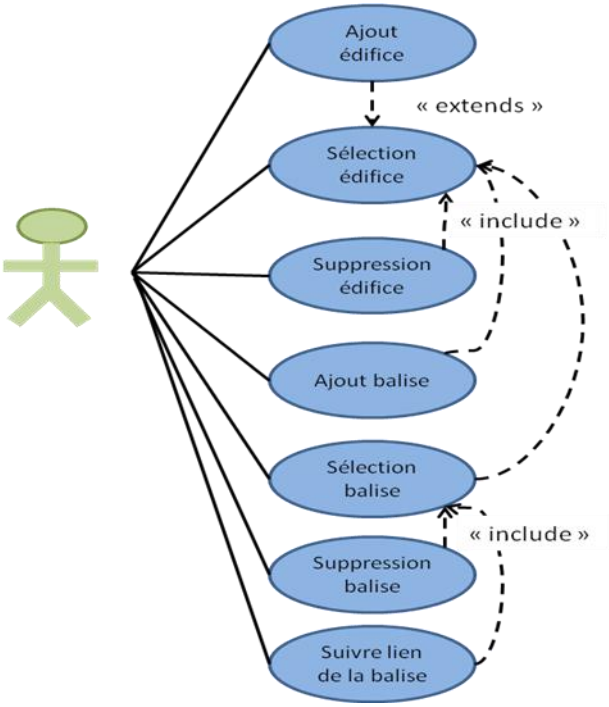

Figure 5 : Cas d'utilisations

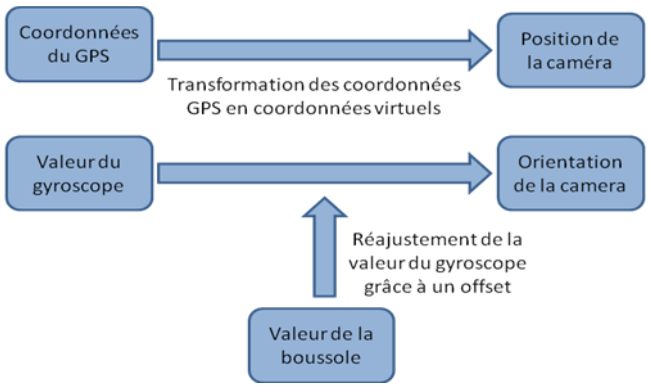

Figure 6 : Interaction automatique des capteurs avec la librairie

4. Structure d'un programme Android

Les concepts fondamentaux pour une bonne compréhension de l'environnement Android sont le *manifest*, les activités, les *Views* et les *intents*. Toutes ces notions ont été essentielles pour la réalisation de notre application et sont abordées ci-dessous.

La programmation sur Android se fait à l'aide de deux langages. L'XML est utilisé de manière systématique pour le *manifest* (couche accès aux données) et la plupart du temps pour définir les *Views* (couche présentation). Java est notre second langage pour réaliser un programme sous Android. Il est utilisé notamment pour décrire les activités et les classes de l'application (couche métier). Il peut aussi définir les *Views.*

Le *manifest* est un fichier inclus dans chaque projet Android au format XML. Il est stocké à la racine du projet – *e.i.* `AndroidManifest.xml`. Il permet de définir la structure et les métadonnées de l'application, ses composants et ses prérequis (Meier, 2010, p. 71). Il inclue des nœuds pour chacun des composants (Activités par exemple) qui constituent l'application. Il détermine les permissions utiles à l'application ainsi que la manière dont les activités interagissent les unes avec les autres.

4.1. Activity – activité

Une activité peut être assimilée à un écran qu'une application propose à son utilisateur. Le passage entre deux écrans correspond au lancement d'une activité ou au retour sur une autre activité placée en arrière-plan. Une application est donc composée d'une superposition d'activités comme le montre la figure 7 ci-contre.

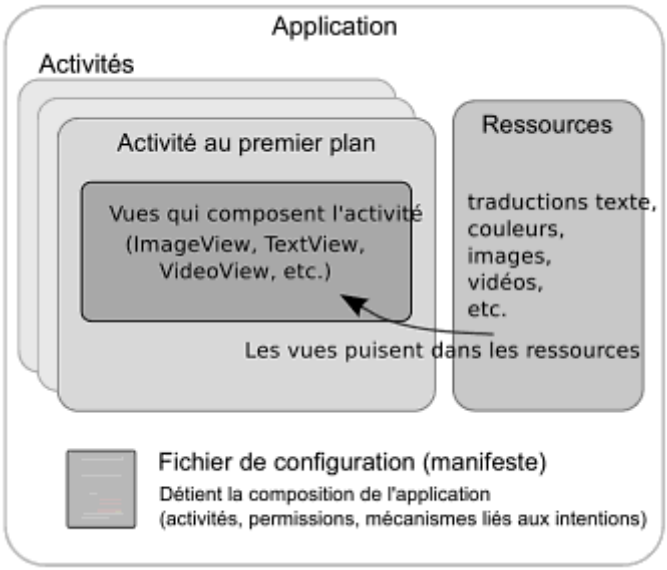

Figure 7 : Composition d'une application (Damien Guignard, 2010, p. 38)

Une activité se décompose en deux parties :

- La partie logique de l'activité et la gestion du cycle de vie de celle-ci sont implémentées en Java dans une classe héritant de `Activity.`
- L'interface utilisateur, qui est le plus souvent défini de façon générale par du code XML placé dans les ressources de l'application.

Cycle de vie d'une activité, les états principaux :

- Active (*active*) : activité visible qui a le focus et interagit avec l'utilisateur. L'appel à la méthode `onResume`, permet de basculer l'activité dans cet état. Elle est ensuite mise en pause quand une autre activité devient active grâce à la méthode `onPause.`
- Suspendue (*paused*) : activité qui n'est pas située au premier plan et qui ne détient pas le focus de l'utilisateur. La méthode `onPause` est invoquée pour entrer dans cet état et les méthodes `onResume` ou `onStop` permettent d'en sortir ;
- Arrêter (*stopped*) : activité non visible. C'est `onStop` qui conduit à cet état.

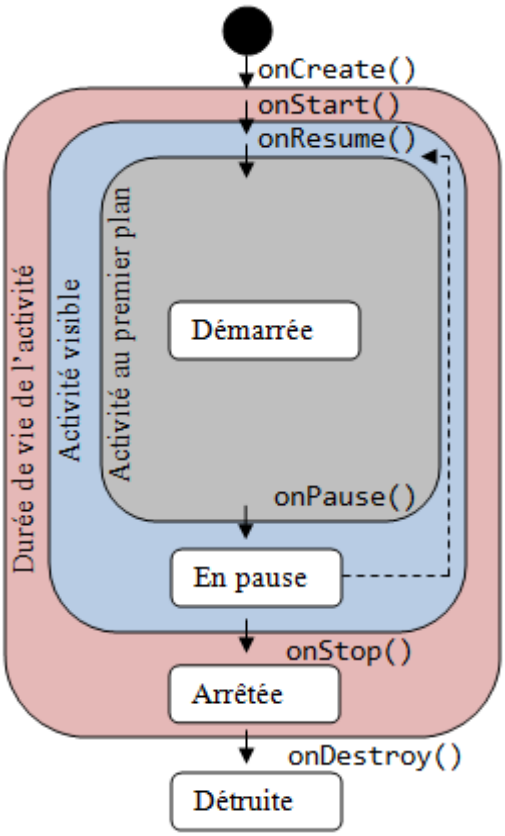

Figure 8 : cycle de vie d'une application (Damien Guignard, 2010, p. 45)

Ci-dessus la figure 8, représente le diagramme d'un cycle de vie d'une d'activité dans ses principaux états ainsi que les différentes transitions. L'application que nous avons réalisée dispose d'une seule activité principale mais bascule vers le navigateur pour afficher les liens des tags.

4.2. Intent – intention

Un *intent* est une fonction permettant de basculer d'une activité à une autre. Ce lien peut être déclaré facilement et permet de passer des paramètres. Les *intents* permettent d'avoir une interaction entre les différentes applications du téléphone et rendent son utilisation homogène tout en tenant compte des choix et préférence de l'utilisateur. Ils ont aussi un rôle primordial pour rendre compatibles les nombreux terminaux Android.

4.3. View – *vues*

Les Views sont des classes de base pour tous les éléments visuels d'interface. Il existe plusieurs View prédéfinis par Android, mais la plupart du temps elles sont définis par le programmeur. Les noms des classes du sdk Android sont choisis pour qu'ils soient le plus explicite. Celle-ci renvoie à la notion de « vue » dans le modèle MVC.

4.4. Drawable

Les *drawables* permettent de normaliser les images utilisées par une application et de limiter les fichiers externes. En effet, les images sont alors converties en objet et simplifient leurs utilisations.

5. La librairie OpenGL

5.1. Présentation

La librairie OpenGL est une API multiplateformes *open source* disponible depuis 1992. Elle dispose d'environ 250 fonctions et permet de créer des scènes en 3 dimensions. . Cette API utilise les principes de la géométrie projective et génère des scènes uniquement composées de lignes, de triangles et de carrés.

Il existe plusieurs versions d'OpenGL répondant à différents besoins. La première est **OpenGL 1.0,** elle utilise un *pipeline* fixe. Cette version manque de modularité mais permet une réalisation rapide. En 2004, **OpenGL 2.0** propose un *pipeline* paramétrable afin de générer uniquement les éléments nécessaires et de se concentrer sur des éléments précis du rendu. Ainsi, la qualité visuelle et la rapidité peuvent être améliorées en dépit de sa mise en place qui est plus complexe.

OpenGL utilise des matrices pour générer les scènes projetées à l'écran et nécessite beaucoup de ressources machine. **OpenGL ES** est une version allégée qui limite les calculs afin de répondre à la puissance limitée des systèmes embarqués. Cette version ne dispose pas de certaines routines de programmation, elle est par exemple plus restrictive sur les primitives et permet seulement la création de triangles.

N'ayant pas de consigne particulière pour le choix de l'API, nous avons opté pour la version ES 1.0. Elle répond à nos besoins et offre une première approche de la réalité virtuelle.

5.2. Les outils

5.2.1. Les objets

5.2.1.1. Les vecteurs

Les vecteurs sont la base de tous les objets OpenGL et délimitent les sommets de celui-ci. Ils sont définis par des coordonnées dans les 3 dimensions (x, y, z) et peuvent être apparentés à des points dans cet espace.

5.2.1.2. Les indices

Les indices indiquent quels vecteurs vont être reliés entre eux. Dans la version ES on associe uniquement des triplets afin de former des triangles. Etant donné que l'on peut voir une seule face à la fois, l'ordre horaire ou antihoraire des vecteurs indique quelle face va être affichée (cf. Figure 9).

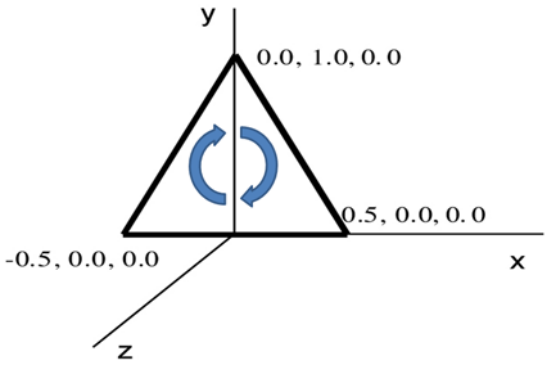

Figure 9 : exemple de vecteurs reliés dans le sens horaire

5.2.2. La classe *Renderer*

Cette classe implémente l'interface *Renderer* et permet d'afficher la scène dans une *SurfaceView*. Elle hérite de trois méthodes permettant d'effectuer le rendu graphique. *OnSurfaceCreated* est appelée à la création du rendu et permet d'initialiser la *SurfaceView*. *OnSurfaceChanged* est appelée quand les dimensions de la fenêtre de visualisation sont modifiées. Enfin, la méthode *onDrawFrame* rafraîchit la scène le plus fréquemment possible. Elle permet d'afficher les différents objets et de faire le lien avec la surface de visualisation.

5.3. Détail des transformations

Les transformations correspondent à la succession d'opérations permettant de passer des vecteurs OpenGL à la scène en 3 dimensions visualisable à l'écran. Nous ferons souvent des analogies avec le maniement d'un appareil photo pour faciliter la compréhension (OpenGL 1.4 guide officiel page 101).

Au cours de la réalisation d'une scène avec OpenGL il faut effectuer des opérations sur les différentes transformations pour obtenir l'effet voulu.

5.3.1. Visualisation-modélisation

Cette première étape relie les vecteurs afin de créer un objet. On situe l'objectif de la caméra ainsi que les objets dans la scène.

5.3.2. Projection

La projection consiste à définir le type de perspective à appliquer. La perspective cavalière ne tient pas compte de la profondeur et « colle » les objets à l'écran. La perspective conique rétrécit les objets plus ils sont éloignés de l'objectif. C'est la perspective la plus souvent utilisée afin d'avoir un effet de « monde virtuel ». Cette transformation permet aussi de définir la taille du champ de vision et les dimensions de l'espace visualisable.

5.3.3. Cadrage

Le cadrage indique la forme sur laquelle la scène est appliquée. Il permet ainsi de savoir ce qui se situe à l'intérieur et à l'extérieur de la scène et de dessiner uniquement les objets visibles.

5.4. Le pipeline

Le pipeline définit les opérations de bas niveaux à effectuer pour le rendu de la scène. Nous avons assez peu travaillé cet aspect car il est géré automatiquement par OpenGL 1.0. Il est cependant intéressant d'avoir connaissance de son fonctionnement pour mieux comprendre certaines méthodes.

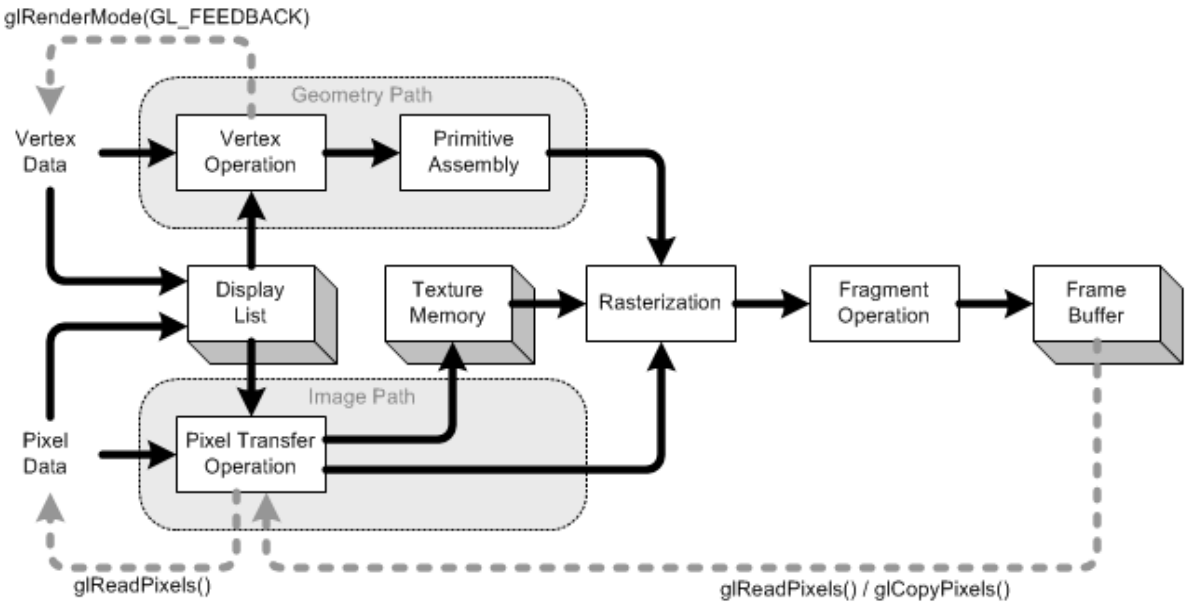

Figure 10 : succession des opérations du pipeline

6. Notre librairie

6.1. Travail effectué

Afin de prendre en main la librairie OpenGL, nous avons cherché à créer un objet et à le visualiser. Ce « Hello world » OpenGL donne un bon aperçu des fonctionnalités de la librairie et des opérations à effectuer pour créer l'environnement virtuel.

Pour nous placer directement dans le contexte de notre travail, nous avons superposé la surface affichant les objets virtuels avec la surface contenant le flux de la caméra. Il a fallu donc au préalable rendre le fond de la surfaceView OpenGL transparent pour obtenir l'effet voulu. Ces opérations s'effectuent dans l'activity. Android propose d'utiliser les View pour offrir ces services.

Ce travail n'est pas complexe mais nécessite beaucoup de pré requis par rapport aux environnements virtuels. Une fois que nous avons pris en main ces aspects, nous avons pu ajouter des fonctionnalités à nos objets.

6.1.1. Sélectionner un objet

La sélection d'objets, aussi appelée picking, n'est pas implémentée de base dans OpenGL. Nous avons donc du concevoir nos propres méthodes pour répondre à ce besoin.

Il existe deux méthodes permettant de connaître l'objet cliqué. La première consiste à lire le pixel affiché à l'écran. Cette technique nécessite de redessiner toute la scène en attribuant une couleur à chaque objet. On compare ensuite la couleur du pixel lu avec celle des objets. Nous n'avons pas retenu cette solution pour deux raisons :

- ✓ La lecture du pixel nous semblait compliquée car pour obtenir cette valeur, il faut aller lire le buffer de l'affichage. Cette opération de bas-niveau nécessite une certaine maîtrise du pipeline OpenGL. De plus, Android utilise d'avantage la notion de « densité de pixel ». En effet devant la diversité des supports et donc de la taille des écrans et de leurs résolutions, il est préférable de normaliser l'affichage afin d'obtenir des images de même taille sur tous les terminaux.

- ✓ Comme expliqué ci-dessus, il faut redessiner la scène à chaque clique. Même si cette scène n'est pas forcement affichée à l'écran, nous avons préféré préserver les ressources système pour ne pas surcharger l'application et la rendre utilisable sur le plus de supports possibles.

OpenGL effectue des projections d'objets en 3 dimensions sur un écran en 2 dimensions. Pour utiliser la deuxième technique de sélection, il faut réaliser l'opération inverse et faire correspondre un point sur l'écran à une coordonnée dans l'espace virtuel. La librairie GLU propose des outils pour OpenGL et notamment la fonction unProject(…). Cette méthode nous renvoie les coordonnées x, y et z du point touché à l'écran. Cependant, tout point en 2 dimensions correspond à une droite en 3 dimensions. La fonction nous renvoie donc toujours à une profondeur z égale à 1 afin d'établir une droite. L'objet cliqué est alors sur la droite passant par la position de la caméra et le point retourné par la fonction.

6.1.1.1. Rayon du picking

La méthode unProject ne tient pas compte de la position de la caméra lors du clique. En effet, elle renvoie toujours z=1. Les coordonnées du point projeté dans l'espace en 3 dimensions sont donc valables uniquement lorsque la caméra est située en (0, 0, 0) et qu'elle est orientée vers le point (0, 0,1). Il faut donc effectuer une translation et une rotation du repère de la projection réalisée par unProject.

6.1.1.1.1. Rotation et translation

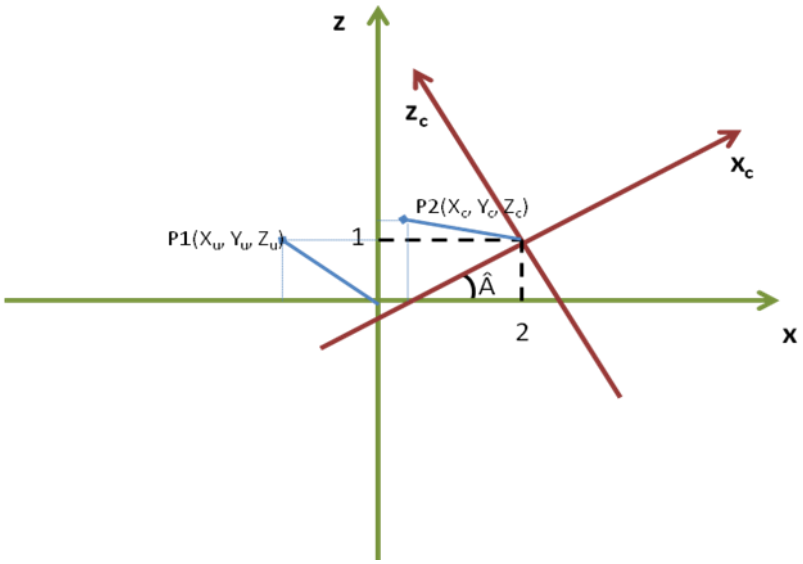

Figure 11 : projection des repères de la caméra et de la méthode unProject() dans le plan x, z.

Légende :

Le plan rouge correspond au repère de la caméra.

Le plan vert correspond au repère de la méthode unProject().

La caméra est située au point (1, 0, 2) et a effectué une rotation d'angle Â.

Le point P1 correspond à la valeur retournée par la méthode unProject().

Le point P2 est la coordonnée que l'on cherche pour créer le rayon.

Nous utilisons les formules de trigonométrie pour effectuer une rotation d'angle Â au point P1.

Xc = Xu*cos(A) + Zu*sin(A)

Zc = Zu*cos(A) - Xu*sin(A)

La translation s'effectue en modifiant le deuxième point de la droite par la position de la caméra (point (1, 0, 2) dans l'exemple).

Nous avons donc les deux points de notre droite nous permettant de tracer le rayon du picking. Cependant, OpenGL ne nous informe pas de l'espace occupé par un objet. Il faut donc définir une enveloppe pour chaque objet permettant de savoir quand celui-ci est traversé par le rayon.

6.1.1.2. Enveloppe

La définition de l'enveloppe se décompose en deux temps. On définit d'abord les points extrêmes de l'objet puis on ajuste en fonction de sa position et de sa rotation.

6.1.1.2.1. Enveloppe relative

Cette première enveloppe est définie lors de la création de l'objet. On parcourt ses vecteurs et on retient les valeurs extrêmes dans les trois plans. Cette méthode définit donc un parallepipède autour de l'objet. Cependant cette enveloppe ne tient pas compte de la position et de la rotation de l'objet par rapport à la caméra.

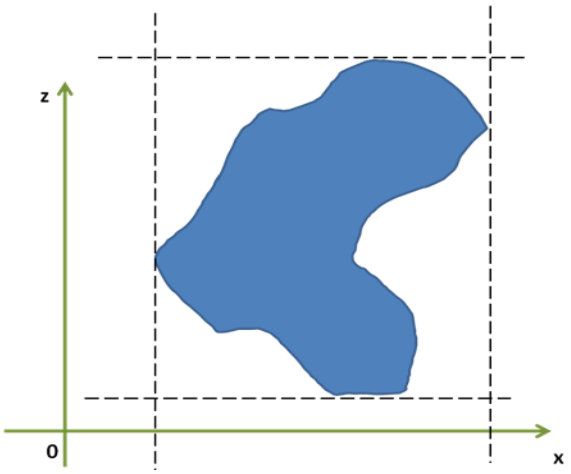

Figure 12 : enveloppe d'un objet dans le repère de la caméra

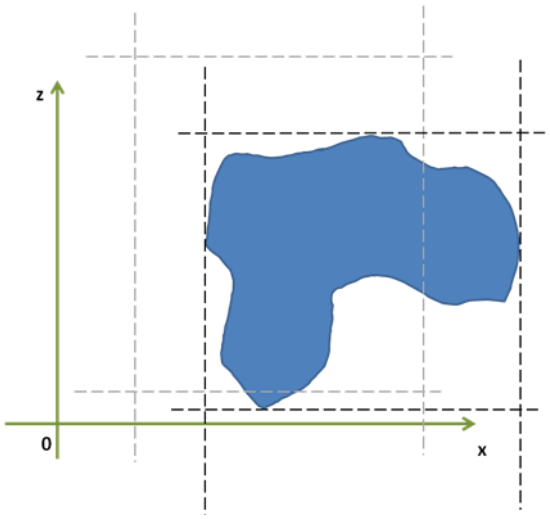

Figure 13 : décalage de l'enveloppe du même objet en fonction de sa position et de sa rotation

6.1.1.2.2. Enveloppe absolue

Cette enveloppe est définie lors d'un clique sur l'écran. On ajuste l'enveloppe relative pour correspondre à la réelle position de l'objet. On utilise, ici aussi, des formules de trigonométrie pour faire la translation et la rotation. Cette décomposition en deux temps pour la création de l'enveloppe permet d'économiser des ressources en n'effectuant pas deux fois les mêmes opérations.

6.1.1.2.3. Limites

Ce type d'enveloppe présente une certaine limite. Celle-ci ne « colle » pas strictement à l'objet. Cet aspect peut faciliter la sélection d'un objet. En revanche cette souplesse peut causer certains problèmes pour les objets concaves. Ceux-ci peuvent entraîner une mauvaise sélection et dérouter l'utilisateur final.

Notre modèle d'enveloppe ne tient pas compte des transformations d'échelle des objets car nous n'effectuons jamais ce type d'opération et que cet aspect n'a jamais été évoqué lors de l'élaboration du cahier des charges.

Enfin, il faut nécessairement que les objets créés soient symétriques dans le plan x, z. C'est-à-dire qu'il faut autant de volume de part et d'autre de l'axe x ainsi que, et de manière indépendante, de part et d'autre de l'axe z. Cette obligation facilite grandement les calculs en fixant le centre de l'objet au centre du plan et permet un calcul rapide de ses diagonales. Ce choix impute relativement peu les libertés de l'utilisateur.

6.1.1.3. Intersection

L'intersection est la dernière étape du picking, elle consiste à tester si le rayon croise une enveloppe. On définit alors un delta qui nous permet de découper la droite. Ce pas doit être assez grand pour permettre de parcourir la droite en un temps raisonnable et assez petit pour ne pas omettre certain point. Ensuite, on teste chaque point pour savoir s'il est compris à l'intérieur d'une enveloppe.

Cette technique fonctionne seulement si les objets et leurs enveloppes ont un volume et sont donc en 3 dimensions.

6.1.2.Déplacer un objet

Nous avons élaboré un moyen de déplacer les objets virtuels afin de les positionner au gré de l'utilisateur. A cette étape du projet, notre librairie ne tient quasiment pas compte de l'axe vertical, nous avons donc décidé de déplacer l'objet sur les axes x et z. C'est-à-dire qu'un mouvement vers le haut de l'écran déplacera l'objet d'avantage en profondeur et *vice versa*.

6.1.2.1. Technique

Notre technique de déplacement est assez similaire à la sélection. En effet, nous projetons un rayon dans l'espace virtuel et le parcourons jusqu'à dépasser un certain plan. Dans notre cas, ce plan est parallèle à l'axe *y* afin de fixer cette valeur. Lorsque le rayon dépasse ce plan, nous interrompons le parcours du rayon et positionnons l'objet aux coordonnées du dernier point du rayon.

6.1.2.2. Limite

Cette technique présente plusieurs défauts qui devront être corrigés.

- ✓ Le rayon n'intercepte le plan que dans la moitié de l'écran. L'utilisateur ne peut donc pas utiliser toute la surface de l'écran.
- ✓ L'utilisateur ne peut pas déplacer l'objet sur les 3 axes car il faudrait changer d'axe de référence. De plus, on ne peut pas effectuer de rotation sur l'objet. Pour palier à ces problèmes, il faudrait entrer dans un autre mode permettant d'effectuer les déplacements voulus.

6.1.3.Position et rotation automatiques des balises

Les balises liées à un édifice doivent être affichées lors d'un clic sur ce dernier. L'utilisateur pouvant être n'importe où autour du bâtiment, il faut positionner les balises de manière à les rendre visibles. Nous avons décomposé ce travail en deux étapes.

6.1.3.1. Position relative du tableau

Lors de la création de la balise, une place lui est attribuée dans un tableau relatif. Ce tableau associe 14 emplacements avec des coordonnées (x, y, z). Ce tableau ne tenant pas compte de la profondeur, z sera toujours égal à 0.

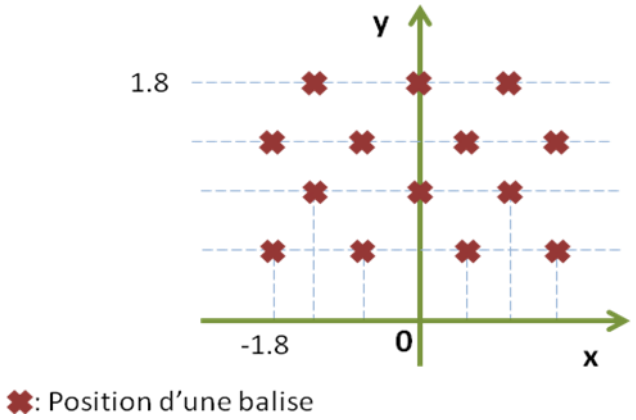

Figure 14 : disposition des 14 balises

6.1.3.2. Position absolue du tableau

Pour positionner les balises correctement, il faut déterminer la position du tableau relatif dans le repère OpenGL. Le plan contenant les balises est toujours situé à la même distance de l'édifice (radius). Nous utilisons les formules de trigonométrie afin de déterminer les positions Z_b et X_b d'une balise. Enfin, la rotation d'une balise est simplement égale à l'angle entre la caméra et l'origine du repère.

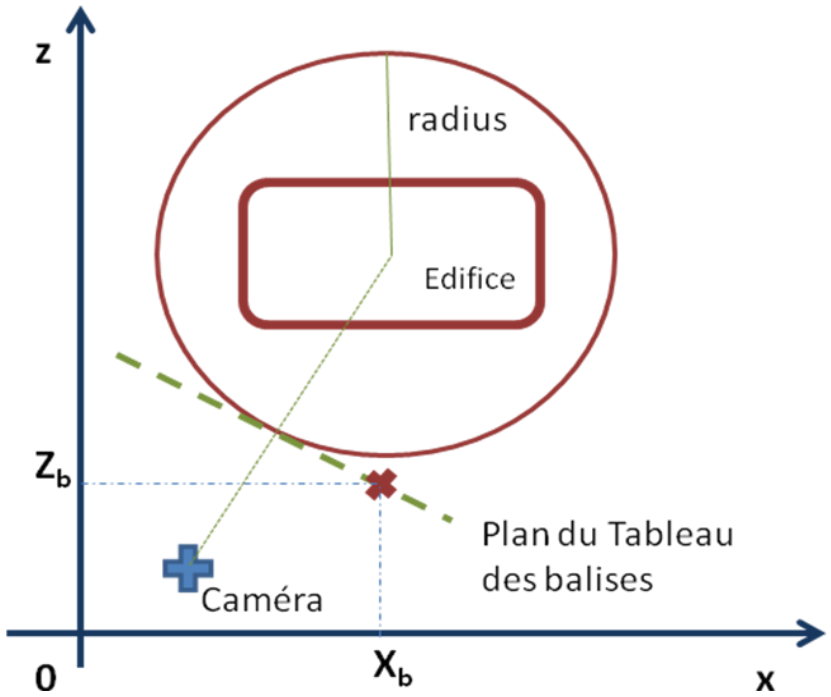

Figure 15 : construction géométrique pour déterminer la position des balises.

6.1.4. Interactions des capteurs avec OpenGL – déplacement de la camera

Les capteurs permettent de déplacer la caméra OpenGL en fonction de la position et de l'orientation du téléphone. Cette concordance permet de donner l'effet de réalité augmentée géo-localisée. Le déplacement longitudinal de la caméra est en lien avec les valeurs du GPS. Le déplacement angulaire est relié à la boussole et au gyroscope.

Il faut transformer les valeurs du GPS pour les faire correspondre avec les dimensions du monde virtuel. Nous avons fait le choix de multiplier la longitude et la latitude par 10000. Ce coefficient permet de faire correspondre 1 unité OpenGL avec environ 10 mètres dans la réalité.

6.2. Utilisation de la librairie

Notre librairie propose les fonctionnalités suivantes aux programmeurs:

- ✓ Création d'un édifice par défaut ou de son choix
- ✓ Création d'un tag par défaut ou de son choix
- ✓ Ajouter un tag à un édifice
- ✓ Mise à jour de l'affichage OpenGL
- ✓ Obtenir la position GPS
- ✓ Obtenir la position GPS devant le téléphone
- ✓ Obtenir la liste des objets affichés
- ✓ Obtenir l'objet sélectionné
- ✓ Déplacer un objet

Pour de plus amples renseignement sur les fonctionnalités de la librairie, nous vous invitons à consulter la javadoc disponible en annexe.

7. Interface de l'application

L'application qui a été réalisée n'est pas une application à but commercial, mais une démonstration de l'utilisation de la librairie. En effet le travail demandé porte sur la réalisation de la librairie. L'application produite nous a servi à vérifier le bon fonctionnement de celle-ci. De plus des parties du code concernant le travail dont nous n'avons pas pris part (e.i. correction des bruit des capteurs et amélioration de la représentation des objets virtuels.) ne sont pas à notre disposition. C'est pour cela que l'application que nous proposons reste un prototype. D'où l'apparition de certains bugs qui sont dus à des problèmes de programmation plus profonds concernant Android ou de code manquant.

L'interface de l'application présente le monde virtuel avec ces objets 3D superposés sur le flux de la caméra. Ainsi que l'implémentation d'un menu principal pour ajouter des bâtiments et un menu contextuel pour effectuer plusieurs options sur un bâtiment sélectionné.

Un story board de l'application est disponible en annexe (cf. annexe 2). Le scénario proposé est comme suit : l'utilisateur lance l'application et voit une scène vide. En appuyant sur le bouton menu du mobile il ajoute un édifice devant lui. Ensuite il ajoute quatre tags puis effectue une rotation sur la

droite pour ajouter un autre bâtiment en lui ajoutant deux tags. Enfin, il décide de cliquer sur le premier édifice pour poster un commentaire sur Facebook.

8. L'organisation du projet

8.1. Méthodologie

Pour le projet, nous avons travaillé à domicile. Un rendez-vous était prévu avec notre maître de stage toutes les semaines, généralement le mardi matin à 10h. Nous parlions du travail réalisé dans la semaine et des objectifs à réaliser pour la semaine d'après. Suivant le travail à produire, le rendez-vous pouvait être décalé d'une semaine. Pour la programmation, nous travaillions sur nos ordinateurs et Smartphone respectifs.

8.2. Travail d'équipe

Par rapport à l'ampleur du projet, nous avons surtout travaillé avec notre maître de stage Betül. Nous avons rencontré une bonne partie de l'équipe, notamment les cadres, Gensel Jérôme et Tellez Bruno, mais nous avons trouvé dommage de ne pas pouvoir rencontrer d'autres exécutants[2] pour pouvoir discuter du projet. Ces rencontres auraient pu avoir des répercussions positives sur notre travail.

8.3. Définition des objectifs

Les objectifs et le cadre de travail on été mal définis en amont, ce qui s'est répercuté sur l'ensemble de notre travail. C'est tout particulièrement avant la période de temps plein que les objectifs on changés. Ce changement de programme est récapitulé ci-dessous.

Les premiers objectifs pour la demande de la librairie :

- Implémentation des capteurs
- Evolution des capteurs avec un objet 3D – *assimilé seulement à un tag*
- Géo localiser un objet 3D
- Corriger les bruits des capteurs[3]

[2] Groupe de deux étudiants Canadien venus en France, sur Lyon, pour travailler sur la correction des capteurs.
[3] Travail abandonné lors des nouveaux objectifs car nous avons appris à ce moment là que des étudiants Canadien avaient travaillés dessus.

Les changements après coup (avant la période de temps plein) :

- Arrêt du travail en cours sur les capteurs (correction des capteurs)
- Implémentation de deux types d'objets 3D (édifice, tag) au lieu d'un tag
- Interaction de l'utilisateur avec l'environnement virtuel – *sélection d'un objet*
- Implémentation de fonction tournant autour de la sélection d'un objet

Objectivement, ce changement de programme ne devait pas nous perturber outre mesure si nous avions étés au courant plus tôt du réel projet dans lequel nous avons pris part (ARCAMA -3D). Cette divergence (ou précision) d'objectif nous à freinés dans la productivité de notre travail. Etant donné le déroulement du TER, ce changement n'était vraiment pas arrivé au moment opportun. Nous avons perdu du temps à trouver des informations pertinentes sur OpenGL ES, cette librairie n'est pas beaucoup documentée sur le web et absente de nos bibliothèques. La recherche plus profonde sur cette librairie aurait pu être anticipée et nous n'aurions pas eu besoin de faire une refonte de notre librairie.

Les causes de ce problème viennent sans doute d'une mauvaise communication entre les deux parties. Nous pensons que nous sommes restés trop passifs dans les phases en amont. Nous aurions dû poser plus de questions à propos du projet Betül. De même, nous aurions du rebondir plus souvent sur ces réponses dans le but d'éclaircir certaines situations. Néanmoins, malgré ces discordes, le déroulement du projet s'est fait correctement.

8.4. Prévision

Etant données les circonstances, nous ne pouvons pas avoir de confrontation directe entre le planning prévisionnel et le planning réel hormis pour voir l'impact du changement de programme. Cependant le digramme de Gantt du planning effectif est disponible (cf. annexe 3).

9. Conclusion

Au travers de ce projet, nous avons pu aborder les aspects pratiques et professionnels de notre formation; une certaine complexité s'est alors dégagée pour utiliser et coordonner les savoirs théoriques enseignés. Une des problématiques de ce projet à été d'intégrer les outils à notre disposition pour en comprendre leurs pertinences et leurs utilités. Cette accaparation à été satisfaisante dans plusieurs cas et notamment au niveau de la technique, en revanche la partie méthodologique a posé plus de problèmes.

La partie technique du projet a nécessité d'apprendre les langages relatifs à Android et à OpenGL. Ces thèmes n'ont pas été abordés en formation et nous avons dû trouver nos propres sources et références. Cette démarche nous a confrontés soit à un surplus soit à un manque d'informations. Dans les deux cas la difficulté est grande et le temps nécessaire important. Face à ce manque de connaissances, il a fallut garder une vision globale du travail pour savoir s'arrêter de rechercher et répondre au problème avec les données disponibles. En effet, toutes les solutions ne sont pas déjà écrites et il faut créer des algorithmes répondant précisément à notre situation. Cette démarche a permis de nous montrer une réalité potentielle du monde professionnel.

L'aspect méthodologique a été beaucoup plus problématique. Face à une méconnaissance de son importance et des impacts sur le projet, il a malheureusement été parfois négligé. De plus, la différence de délai entre notre projet et celui de notre tutrice aurait pu d'avantage nous alerter et nous aurions dû mieux définir et clarifier notre rôle. Les répercutions ont donc été nombreuses et étalées tout au long de notre travail. Ce travail faisait peut-être appel à trop de concepts nouveaux à notre égard pour nous permettre d'avoir une visibilité correcte sur le projet.

De manière transversale, la communication et la compréhension ont donc une importance particulière dans la réalisation d'un travail collaboratif. Nous avons compris l'intérêt d'utiliser des outils de conception et de normalisation du langage afin de bien délimiter le sujet et le travail à réaliser.

Cette première approche de la réalité augmentée a mis en lumière les possibilités ainsi que les limites de cette technologie. La gestion et la précision des capteurs apparaissent donc comme déterminant pour permettre le meilleur effet.

Ce projet a donc été pour nous une excellente expérience et nous a permis de nous confronter à nos idées préconçues sur la réalisation d'un projet. Face à ces erreurs classiques, nous allons pouvoir aborder avec un nouveau regard le monde professionnel et notamment la réalisation de notre stage pratique en master 2.

10. Webographie

Object/Object Intersection http://www.realtimerendering.com/intersections.html

Tutoriel OpenGL ES 3 - Coloriage et textures http://ipup.fr/page.php?id=265

Texture Mapping – OpenGL Android (Displaying Images using OpenGL and Squares)
http://obviam.net/index.php/texture-mapping-opengl-android-displaying-images-using-opengl-and-squares/

gluUnProject ? Picking - Drag [Résolu] - Forum des professionnels en informatique
http://www.developpez.net/forums/d559439/applications/developpement-2d-3d-jeux/api-graphiques/opengl/gluunproject-picking-drag/

trans.pdf (Objet application/pdf) http://beru.univ-brest.fr/~singhoff/ENS/UE_Animation/CM/trans.pdf

47 Free High Quality Building Textures | RockThe3D http://www.rockthe3d.com/47-free-high-quality-building-textures/

Api Wikitude | Android-France http://android-france.fr/tag/api-wikitude/

exemple de réalité augmentée sur Android http://developpeur.orange.tn/developper-ensemble/tutoriels/tutoriels/exemple-de-realite-augmentee-sur-android

Wikitude Wikitude.com

Rapport projet Android.pdf (Objet application/pdf)
http://www.isima.fr/~lacomme/L3/doc/Rapport%20projet%20Android.pdf

OpenGL ES Tutorial for Android – Part IV – Adding colors | Jayway Team Blog - Sharing Experience
http://blog.jayway.com/2010/01/14/opengl-es-tutorial-for-android-%e2%80%93-part-iv-adding-colors/

OpenGL ES with Android Tutorial- Switching from Canvas to OpenGL
http://obviam.net/index.php/opengl-es-with-android-switching-from-canvas-to-opengl/

19. OpenGL ES - YouTube http://www.youtube.com/watch?v=_WcMe4Yj0NM&feature=related

All about OpenGL ES 2.x – (part 1/3) http://db-in.com/blog/2011/01/all-about-opengl-es-2-x-part-13/

Android 3D with OpenGL ES
http://www3.ntu.edu.sg/home/ehchua/programming/android/Android_3D.html

11. Bibliographie

Astle, D. (2004) *OpenGL ES Game Development*. Thomson

Betül Aydın, Jérôme Gensel, Sylvie Calabretto and Bruno Tellez "ARCAMA-3D - A Context-Aware Augmented Reality Mobile Platform for Environmental Discovery", *In Proceedings of the 11th International Symposium on Web and Wireless Geographical Information Systems: W2GIS'12*, Di Martino, S., Peron, A., Tezuka, T. (eds.), pp. 17,26, Springer LNCS 7236, Naples, Italy, April 12-13, 2012. (***short paper***)

Betül Aydın, Jérôme Gensel, Sylvie Calabretto et Bruno Tellez "ARCAMA-3D: une application mêlant réalité augmentée et 3D pour la présentation d'informations contextuelles", Dans les actes de 8èmes journées francophones Mobilité et Ubiquité (UBIMOB'12), Anglet, France, pages 34-41, June 2012.

Guignard, D., & Chable, J., & Robles E. (2010). *Programmation Android.* Paris: EYROLLES.

Mottier, C., & Perrier, L. (2011). *Dévelloper pour Android.* Norderstedt: Digit Books.

Murphy, M. (2011). *L'art du développement Android*. Paris: Pearson.

Smithwick, M. (2012). *Pro OpenGL ES for Android*.Apress

Woo, M. (2004). *OpenGL 1.4 guide official*. Paris: CampusPress.

Zeckner, M. (2010). *Beginning OpenGL Game Programming*.Apress

12.Annexes

1. Javadoc

a. Package com.android.augRea

Class Summary	
Class	**Description**
Balise	Cette classe permet de gérer les objets OpenGL de type balises
CameraPreview	A basic Camera preview class
Edifice	Cette classe permet de gérer les objets OpenGL de type edifice
Espace_Virtuel	Cette classe est l'interface de la librairie elle permet de modifier des balises, de se déplacer, de fournir la surfaceview (renderer)
EVrenderer	cette classe permet d'afficher les objets openGL
Gps	Cette classe permet de gérer le capteur GPS
Movement	cette classe permet de connaitre les valeurs des differents capteurs et de modifier la vue opengl en fonction
ObjectOGL	Cette classe permet de créer des balises Ces balises sont stocker dans une List de Espace_Virtuel

b. Class Balise

- java.lang.Object
  - com.android.augRea.ObjectOGL
    - com.android.augRea.Balise

```
public class Balise
extends ObjectOGL
```

Cette classe permet de gérer les objets OpenGL de type balises

Author:

Aouizerate Erik & Faucher Julien

Field Summary

Fields	
Modifier and Type	**Field and Description**
`private static byte[]`	**`indices`** indice par defaut des balises
`private static float[]`	**`texture`** positon par defaut des textures
`java.lang.String`	**`url`** lien de la balise
`private static float[]`	**`vertices`** vecteur par defaut des balises

Fields inherited from class com.android.augRea.ObjectOGL

coordonnee, eV, textureDejaCharge

- **Constructor Summary**

Constructors
Constructor and Description
Balise(**Espace_Virtuel** eV, float x, float y, float z, float[] vertices, byte[] indices, int drawable, float[] texture) constructeur
Balise(**Espace_Virtuel** eV, float x, float y, float z, int drawable) constructeur

- **Method Summary**

- **Methods inherited from class com.android.augRea.ObjectOGL**

draw, getCoordonee, getEnvelope, getRotate, getVisible, loadGLTexture, setCoordonnee, setRotate, setVisible

- **Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

c. Class CameraPreview

- java.lang.Object
  - SurfaceView
    - com.android.augRea.CameraPreview

```
public class CameraPreview

extends SurfaceView
```

A basic Camera preview class

Field Summary

Fields	
Modifier and Type	**Field and Description**
`private Camera`	**`mCamera`**
`private SurfaceHolder`	**`mHolder`**
`private static java.lang.String`	**`TAG`**

Constructor Summary

Constructors	
Constructor and Description	
`CameraPreview``(Context context, Camera camera)`	

Method Summary

Methods	
Modifier and Type	**Method and Description**
void	**surfaceChanged**(SurfaceHolder holder, int format, int w, int h)
void	**surfaceCreated**(SurfaceHolder holder)
void	**surfaceDestroyed**(SurfaceHolder holder)

- **Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

d. Class Edifice

- java.lang.Object
  - com.android.augRea.ObjectOGL
    - com.android.augRea.Edifice

```
public class Edifice
extends ObjectOGL
```

Cette classe permet de gérer les objets OpenGL de type edifice

Author:

Aouizerate Erik & Faucher Julien

Field Summary

Fields

Modifier and Type	Field and Description
`private java.util.List<Balise>`	**baliseList** liste des balises associés à l'objet
`private static byte[]`	**indices** les indices par defaut d'un edifice
`private float`	**radius** distance entre l'objet et les balises
`private float[][]`	**slotBalise** emplacement relatif des balises

private static float[]	**texture** les indices de la texture
private static float[]	**vertices** les vecteurs par defaut d'un edifice

- **Fields inherited from class com.android.augRea.ObjectOGL**

coordonnee, eV, textureDejaCharge

- **Constructor Summary**

Constructors

Constructor and Description
Edifice(**Espace_Virtuel** eV, float x, float y, float z, float[] vertices, byte[] indices, int drawable, float[] texture)
Edifice(**Espace_Virtuel** eV, float x, float y, float z, int drawable) constructeur

- **Method Summary**

Methods

Modifier and Type	Method and Description
boolean	**addBalise**(**Balise** b) permet d'ajouter une balise à l'edifice
java.util.List<**Balise**>	**getBaliseList**() retourne la liste des balises associé à l'édifice
private float[]	**getCoordCentrale**()

	permet d'obtenir les coordonnées d'un point situé entre l'objet et la caméra à la distance du radius
float	**getRadius** () getter du radius
void	**setCoordBalise** () place les balises en fonctions de la position de la caméra
void	**setRadius** (float r) definit la distance entre l'edifice et les balises
private void	**setSlotBalise** () definit la position relative des balises
void	**supprAllBalise** () supprime toutes les balises
void	**supprBalise** (**Balise** b) supprime une balise associé à l'édifice

- **Methods inherited from class com.android.augRea.ObjectOGL**

draw, getCoordonee, getEnvelope, getRotate, getVisible, loadGLTexture, setCoordonnee, setRotate, setVisible

- **Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

-

e. Class Espace_Virtuel

- java.lang.Object
-
- com.android.augRea.Espace_Virtuel

```
public class Espace_Virtuel
extends java.lang.Object
```

Cette classe est l'interface de la librairie elle permet de modifier des balises, de se déplacer, de fournir la surfaceview (renderer)

Author:

Aouizerate Erik & Faucher Julien

Field Summary

Fields	
Modifier and Type	**Field and Description**
com.android.augRea.AugReaActivity	**context** permet d'acceder aux capteurs et à tout autres ressources du telephone
private **Edifice**	**edificeSelect** garde en mémoire l'édifice selectionnée
private **Movement**	**movement** objet permettant d'acceder au capteurs
private java.util.List<**ObjectOGL**>	**objectList** contient la liste des objets visualisable

private **ObjectOGL**	**objectOGLSelect** garde en mémoire l'objet selectionnée
private **EVrenderer**	**renderer** renderer permetant de visualiser les objets en 3D

Constructor Summary

Constructors
Constructor and Description
Espace_Virtuel(com.android.augRea.AugReaActivity context) constructeur

Method Summary

Methods	
Modifier and Type	**Method and Description**
void	**addBaliseToEdifice**(int drawable) ajoute une balise à un edifice
void	**afficher**() notifie le renderer des changements à l'affichage
void	**afficherBalise**() affiche les balises d'un edifice
Balise	**createBalise**(float x, float y, float z, float[] vertices, byte[] indices, int drawable, float[] texture)

	permet de créer une balise avec les dimensions de son choix
Balise	**createBalise**(float x, float y, float z, int drawable) Crée une balise avec les dimensions par défauts
Edifice	**createEdifice**(float x, float y, float z, float[] vertices, byte[] indices, int drawable, float[] texture) permet de créer un edifice avec les dimensions de son choix
Edifice	**createEdifice**(float x, float y, float z, int drawable) Crée un edifice avec les dimensions par défauts
void	**evenementOnLongTouch**(MotionEvent event) gere les evenement pour un clic long
void	**evenementOnTouch**(MotionEvent event) gère les évenements du toucher tactile et appel la bonne fonction
private void	**eventOnLongTouch**(float xWin, float yWin) evement lors d'un clic long
private void	**eventOnTouch**(float xWin,

	float yWin) appeler lors d'un clique sur l'écran
protected float	**getAngleGyro** () retourn l'angle entre la camera etle nord
protected float[]	**getCameraPosition** () getteur de la position de la camera via le renderer
Context	**getContext** ()
protected **Edifice**	**getEdificeSelect** () retourne le dernier edifice selectioné
protected java.util.List<**ObjectOGL**>	**getObjectList** () getter de la liste des objets
float[]	**getPositionDevant** () permet de retourner la position devant soi
EVrenderer	**getRenderer** () getter du renderer
protected void	**moveObject** (float xWin, float yWin) appellé lors d'un deplacement sur l'écran => déplacement d'une balise
private **ObjectOGL**	**objectTouch** (float xWin, float yWin) permet d'effectuer une sélection sur un objet

void	**onPause**()
void	**onResume**()
void	**onStop**()
protected void	**setCameraPosition**(float x, float y, float z) permet de déplacer la caméra à un point donnée
protected void	**setCameraTolook**(float x, float y, float z) permet de déplacer le regard de la caméra à un point donnée
void	**supprimerObject**(**ObjectOGL** o) supprime un objet

- **Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

-

f. Class EVrenderer

- java.lang.Object
  - com.android.augRea.EVrenderer

```
public class EVrenderer
extends java.lang.Object
```

cette classe permet d'afficher les objets openGL

Author:

Aouizerate Erik & Faucher Julien

Field Summary

Fields	
Modifier and Type	**Field and Description**
`private float[]`	**`cameraPosition`**
`private float[]`	**`cameraToLook`**
`private Espace_Virtuel`	**`eV`**
`private boolean`	**`listObjectEdit`** permet de savoir si la listeTemporaire à été modifié
`private static float[]`	**`modelViewMatrix`**
`private java.util.List<ObjectOGL>`	**`objectList`** liste des balises à afficher
`private java.util.List<ObjectOGL>`	**`objectListTemp`**

	liste des nouvelles balises <= on sauvegarde la nouvelle liste et on la mettra à jour pendant une phase non-critique. => evite les accès concurents
private static float[]	**pointInPlane**
private static float[]	**projectionMatrix**
(package private) float	**var**
private static int[]	**viewport**

Constructor Summary

Constructors

Constructor and Description
EVrenderer(**Espace_Virtuel** eV, java.util.List<**ObjectOGL**> o) Constructor on ajoute de base les balises

Method Summary

Methods

Modifier and Type	Method and Description
private void	**editObjectList**(GL10 gl) permet de modifier réellement la liste des balises évite les accès concurents
float[]	**getCameraPosition**() getter de la position de la camera
float[]	**getCameraToLook**() getter de l'endroit ou regarde la camera

void	**loadTexture** (GL10 gl)
void	**notifyObjectList** (java.util.List<**ObjectOGL**> o) pour modifier la List des balises
void	**onDrawFrame** (GL10 gl) methode appeler en boucle pour rafraichir la scène
void	**onSurfaceChanged** (GL10 gl, int width, int height) methode appellée quand la surface change
void	**onSurfaceCreated** (GL10 gl, EGLConfig config) appellé à la création de la surface
void	**setCameraPosition** (float[] pos) modifie la posiiton de la caméra
void	**setCameraToLook** (float[] look) modifie le regard de la camera
float[]	**unProject** (float xWin, float yWin) permet d'effectuer une projection à l'aide GluUnProject (2D -> 3D)

- **Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

g. Class Gps

- java.lang.Object
- com.android.augRea.Gps

```
public class Gps
extends java.lang.Object
```

Cette classe permet de gérer le capteur GPS

Author:

Aouizerate Erik & Faucher Julien

Field Summary

Fields	
Modifier and Type	**Field and Description**
protected double	**altitude**
(package private) Context	**context**
private **Espace_Virtuel**	**eV** espace virtuel
private Location	**gpsLocation** pour accéder aux variable de localisation longitude latitude altitude
protected double	**latitude**
private LocationManager	**locationManager** location manager
protected double	**longitude**

private **Movement**	**movement** pour envoyer les changements de données à la classe movement
private java.lang.String	**typeContext** le contexte du service

Constructor Summary

Constructors
Constructor and Description
Gps(Context context, **Espace_Virtuel** eV, **Movement** movement) constructeur du GPS

Method Summary

Methods	
Modifier and Type	**Method and Description**
double	**getAltitude**()
double	**getLatitude**()
double	**getLongitude**()
void	**initGPS**() initialisation du GPS
void	**updatePosition**(Location gpsLocation) pour update la position

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class Movement

- java.lang.Object
- com.android.augRea.Movement

---

public class **Movement**

extends java.lang.Object

cette classe permet de connaitre les valeurs des differents capteurs et de modifier la vue opengl en fonction

Author:

Aouizerate Erik & Faucher Julien

Field Summary

Fields

Modifier and Type	Field and Description
(package private) SensorEventListener	**accelerometreListenner**
private float	**angle** angle de base attribuer au gyroscope (version deboguage!)
(package private) SensorEventListener	**boussoleListenner**
Context	**context** permet d'acceder aux capteurs

private **Espace_Virtuel**	**eV** permet l'interaction avec l'espace virtuel
private **Gps**	**gps**
(package private) SensorEventListener	**gpsListenner**
(package private) SensorEventListener	**gyroscopeListener**
private float[]	**nord**
private float[]	**position** position de la camera
private Sensor	**sensorAccelerometre**
private Sensor	**sensorBoussole**
private Sensor	**sensorGPS**
private Sensor	**sensorGyro**
private SensorManager	**sensorManager** pour les capteurs ...

- ## Constructor Summary

Constructors
Constructor and Description
Movement(Context context, **Espace_Virtuel** eV) constructeur

- ## Method Summary

Methods

Modifier and Type	Method and Description
double	**getAltitute**()
float	**getAngle**()
double	**getLatitude**()
double	**getLongitude**()
private void	**initAccelerometre**()
private void	**initBoussole**()
private void	**initGyro**() permet de mettre en route le gyro!
void	**onResume**() necessaire pour liberer les capteurs
void	**onStop**()

- **Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

h. Class ObjectOGL

- java.lang.Object
  - com.android.augRea.ObjectOGL
- **Direct Known Subclasses:**

  Balise, Edifice

```
public abstract class ObjectOGL
extends java.lang.Object
```

Cette classe permet de créer des balises Ces balises sont stocker dans une List de Espace_Virtuel

Author:

Aouizerate Erik & Faucher Julien

Field Summary

Fields	
Modifier and Type	**Field and Description**
`private java.nio.ByteBuffer`	**byteBuf** The buffer holding bytes
`protected float[]`	**coordonnee** coordonnée gps {latitude, altitude, longitude}
`private int`	**drawable** correspond à l'image de la texture
`private float[][]`	**envelope** points extrèmes de la balise permettant de définir son envelope (nécessaire pour le picking)

protected **Espace_Virtuel**	**eV**
private java.nio.ByteBuffer	**indexBuffer** The buffer holding the indices
private byte[]	**indices** les indices définissent l'ordre des vecteurs
private float	**rotate**
private float[]	**texture** correspond aux dimensions de la texture à appliquer
private java.nio.FloatBuffer	**textureBuffer**
boolean	**textureDejaCharge**
private int[]	**textures**
private java.nio.FloatBuffer	**vertexBuffer** The buffer holding the vertices
private float[]	**vertices** coordonnées de l'objet OpenGL dans les 3 dimensions
private boolean	**visible** affiche ou non la balise

- **Constructor Summary**

Constructors
Constructor and Description
ObjectOGL(**Espace_Virtuel** eV, float x, float y, float z, float[] vertices, byte[] indices, int drawable, float[] texture)

constructeur

- **Method Summary**

Methods	
Modifier and Type	**Method and Description**
void	**draw** (GL10 gl) permet de dessiner l'objet
float[]	**getCoordonee** () getter des coordonnees
float[][]	**getEnvelope** () défini l'envelope par rapport à la position de la balise et à sa rotation
float	**getRotate** ()
boolean	**getVisible** () getter pour l'attribut visible
void	**loadGLTexture** (GL10 gl, Context context) chergement de la texture
private void	**setBuffer** () initialise les buffers
void	**setCoordonnee** (float[] coord)
private void	**setEnvelope** () défini les contours de la balise codé en dur mais possibilité de l'automatiser

void	**setRotate**(float r)
void	**setVisible**(boolean v) setter pour l'attribut visible

- **Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

-

2. Story board

Figure 1 - écran au lancement de l'application sans objet

Figure 2 - écran quand l'utilisateur clic sur le menu du mobile - *cercle rouge*

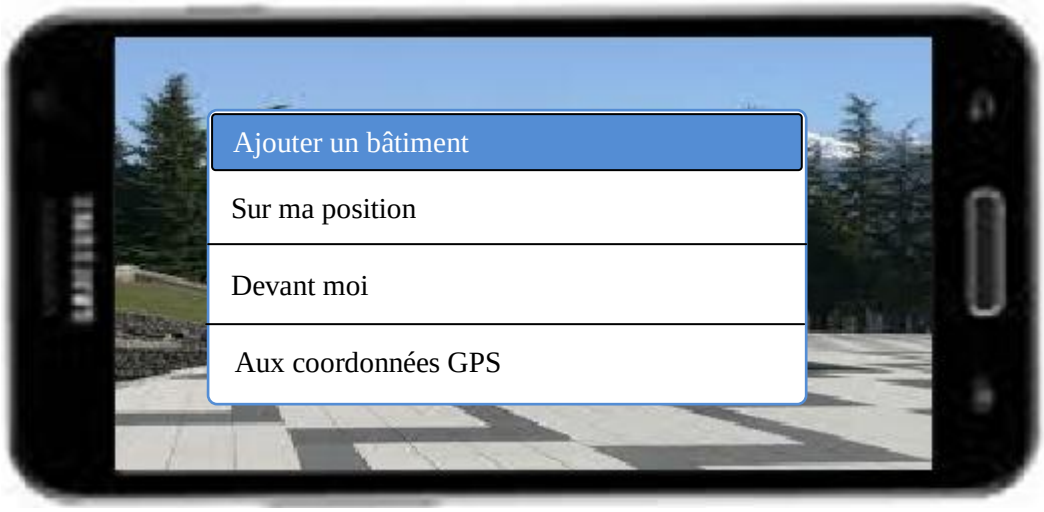

Figure 3 - écran du menu « ajouter un bâtiment »

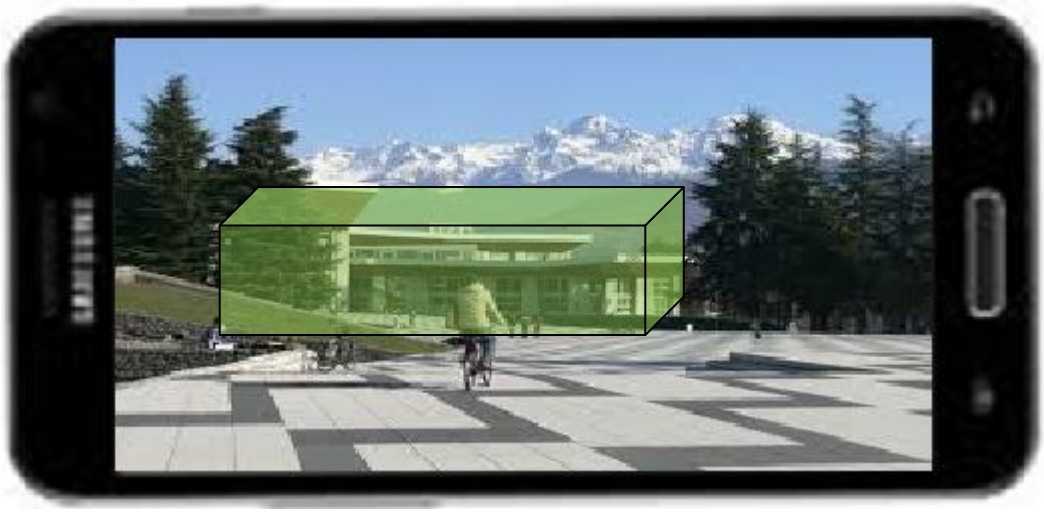

Figure 4 - écran après ajout d'un bâtiment "devant moi"

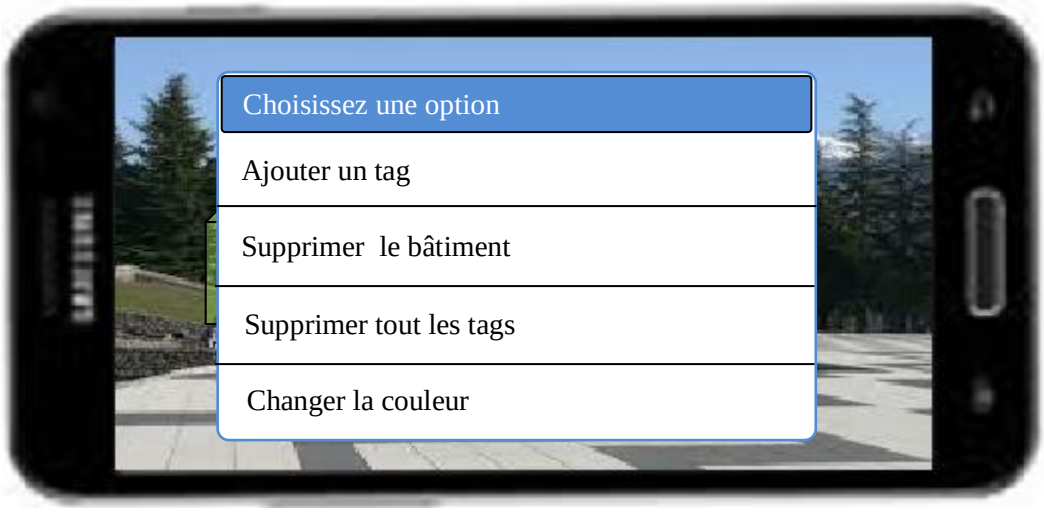

Figure 5 - menu contextuel après un long clic sur un édifice

Figure 6 - écran après l'ajout de quatre tags

Figure 9 - aperçu de l'écran après une légère rotation sur la droite

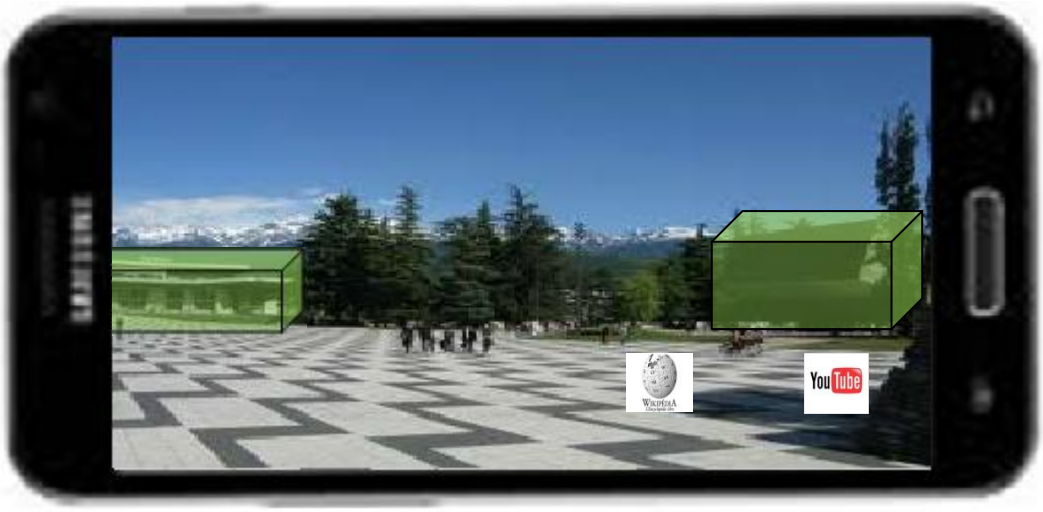

Figure 8 - écran après l'ajout d'un nouveau bâtiment et de deux tags

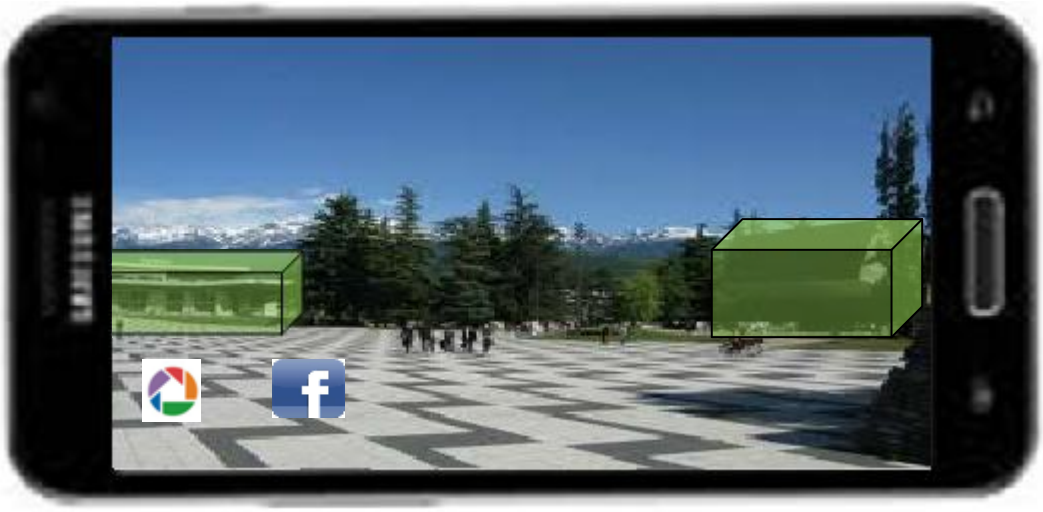

Figure 7 - affichage contextuel des informations multimédia d'un bâtiment lors d'un clic court

Figure 10 - écran après un clic sur l'icône Facebook

3. Planning effectif