



MASTER INFORMATIQUE ISV

RAPPORT DE STAGE

Mise au point d'un capteur 3D par Vision Stéréoscopique

Auteurs:

Yanis MOUNSAMY

Tuteur de stage:

Téophane LANGLAIS

Enseignant référent:

Anne VIALARD

25 août 2016

Remerciement

[...]

Résumé

[...]

Table des matières

| | | |
|----------|--|-----------|
| 1 | Introduction | 4 |
| 2 | Présentation de l'entreprise | 4 |
| 3 | Domaine | 5 |
| 4 | Contexte | 6 |
| 5 | Présentation du sujet | 10 |
| 6 | Travail Effectué | 11 |
| 6.1 | Cadre et méthode de Travail | 11 |
| 6.2 | Etat de l'art des technologies à utilisées | 11 |
| 6.3 | Piste envisagé 1 : Utilisation des caméras 3D | 14 |
| 6.3.1 | Realsense | 15 |
| 6.3.2 | ZED | 18 |
| 6.4 | Piste envisagée 2 : Utilisation des capteurs | 18 |
| 6.4.1 | Objectif | 19 |
| 6.4.2 | Information sur les technologies à utiliser | 19 |
| 6.4.3 | Prise en main des capteurs | 20 |
| 6.4.4 | Création d'une application répondant à l'objectif fixé | 23 |
| 6.5 | Piste envisagé 3 : Utilisation d'Aruco | 25 |
| 6.5.1 | Information sur les technologies à utiliser | 26 |
| 6.5.2 | Prise en main d'Aruco | 26 |
| 6.5.3 | Création d'une application répondant à l'objectif fixé | 27 |
| 6.6 | Bonus : Gestion des occlusion avec la ZED caméra | 28 |
| 7 | Bilan et perspectives | 28 |

1 Introduction

Etudiant en deuxième année de master Informatique dans la spécialité image, son vidéo, parcours 3D et réalité virtuelle, j'ai choisi d'effectuer mon stage de fin d'étude au sein de l'entreprise Logyline, éditrice de logiciels, dont certains exploitent la réalité augmentée (RA). La RA, qui permet d'intégrer des éléments virtuels dans le monde qui nous entoure, est un phénomène émergent depuis quelques années et qui prend de plus en plus d'ampleur, notamment avec le développement de casque tel que l'hololens[3] et magic leap[2], ou la sortie du jeu "Pokemon Go"[4] qui a fait un buzz considérable. J'ai souhaité m'orienter vers ce domaine qui m'attirait et dont j'avais déjà abordé quelques notions dans l'UE Réalité virtuelle de ma formation. J'ai alors rejoins Logyline pour une durée de 5 mois, du 04 avril au 02 septembre, afin d'approfondir mes connaissances et d'enrichir mes compétences en participant à l'amélioration d'un de leur logiciel existant : LOGYConcept3D Pool.

Je présenterai dans un premier temps l'entreprise, puis le domaine de la RA dans lequel se situe mon stage. J'introduirai par la suite le contexte, ce qui permettra de mieux expliquer le sujet. Je poursuivrai par le travail réalisé au sein de l'entreprise où je serai expliqué la méthode de travail appliquée et les différentes difficultés rencontrées. Enfin, je terminerai par un bilan et des perspectives liées à mon sujet.

2 Présentation de l'entreprise

Logyline est une entreprise de logiciels et services informatiques située à Mérignac, créée en 2007 par Théophane Langlais, actuel dirigeant. Niveau service, elle participe à la réalisation de projets événementiels en modélisant des objets de grandes tailles en textile imprimé. En ce qui concerne les logiciels développés, ils sont à destination des professionnels des textiles techniques (voile d'ombrage, bâche) et de la piscine. Ces logiciels ont pour vocation de répondre à des problématiques commerciales (pour aider à la vente) et techniques (réalisation de plan, information de production) ce qui permet aux utilisateurs de gagner en productivité, en fiabilité et de pénétrer de nouveaux marchés. En particulier, leur suite de logiciels "LogyConcept3D" avec LogyConcept3D Pool pour la piscine et LogyConcept3D ShadeSail pour la voile d'ombrage, permettent d'offrir aux clients un aperçu du résultat final avant d'entamer les travaux. Pour ce faire des éléments virtuels (représentation 3D d'objets tels que la piscine et la voile d'ombrage) sont intégrés de façon réaliste sur des photos prises par le client : on parle de réalité augmentée.

3 Domaine

La réalité augmentée est un concept visant à enrichir la perception que nous avons avec le monde réel avec lequel nous interagissons. Cela se manifeste par un ensemble de techniques informatiques permettant d'incrustier des éléments virtuels de façon cohérente au sein d'éléments réels (photographie, vidéo filmée ...), le tout en temps réel. Bien qu'on trouve souvent des cas liés à la perception visuelle, la RA s'applique également à la perception sonore comme c'est le cas par exemple avec une application du CNAM pour les visites de musées par les malvoyants, en produisant un effet sonore à proximité de l'oeuvre exposée [1]. Les applications de réalité augmentée sont diverses et variées et touchent de plus en plus de domaines, tels que le jeux vidéo, la médecine, le tourisme, ou encore le commerce.



FIGURE 1 : Exemple de réalité augmentée avec LogyConcept3D (LC3D) Pool de Logyline

Une des notions importantes de la RA est le fait de pouvoir assurer la cohérence entre la scène virtuelle (ensemble des éléments virtuels) et la scène réelle. En ce qui concerne la suite de logiciel LC3D, cela implique quatres contraintes principales :

- Détection du sol : l'objet virtuel doit être automatiquement posé sur le sol
- Respect des mesures : cohérence entre la taille des objets virtuels avec celle des objets réels figurant sur l'image

- Gestion des occlusions : si l'objet virtuel est positionné de façon à se trouver derrière un élément réel de la photo (mur, arbre, etc) alors le masquage de l'objet doit être visible
- Prise en compte des jeux de lumière (ombres, reflets) entre les différents objets.

C'est autour de cette notion que se déroulera mon stage, il convient alors de savoir comment cela est traité dans les logiciels de Logyline.

4 Contexte

Dans LC3D, la scène virtuelle est gérée grâce à OpenGL, une bibliothèque graphique. Pour mieux comprendre certains termes qui seront abordés par la suite, voici quelques notions de base sur OpenGL :

Les différents objets virtuelles sont représentés par un ensemble de point 3D appelé "sommet". Un sommet est positionné dans l'espace virtuelle à partir d'un repère 3D qui est représenté par un sommet (dit l'origine du repère) et trois vecteur unitaire, caractérisé par un sommet de départ (l'origine), une direction et une longueur (dit la "norme") qui constituera l'unité de base dans la scène (Figure 2).

Les trois vecteur sont représentatif des axes X , Y et Z, pour faire simple, on peut dire dans notre cas que l'axe X est relatif à la gauche ou à la droite (largeur), l'axe Y au fait d'être avancé ou éloigné (profondeur) et l'axe Z au haut ou au bas (hauteur). Ce sont ces trois notion, largeur, hauteur, profondeur qui nous permettent de caractériser ce que nous voyons, c'est-à-dire une vision en 3D. Cela se schématisse comme suit dans un scène virtuelle :

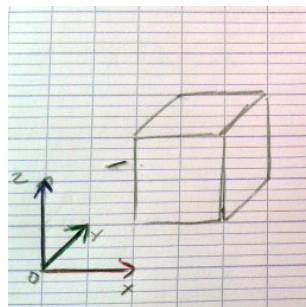


FIGURE 2 : Représentation d'un objet 3D virtuelle par rapport au repère 3D

Ainsi si on choisit de déplacer l'ensemble des sommets caractérisant notre objet dans la direction du vecteur (OX), notre objet se retrouve alors décalé vers la droite.

Le déplacement illustré dans la figure 3 est appelé une transformation. Les transformations principales nous permettront de déplacer nos objets d'une position à une autre en suivant

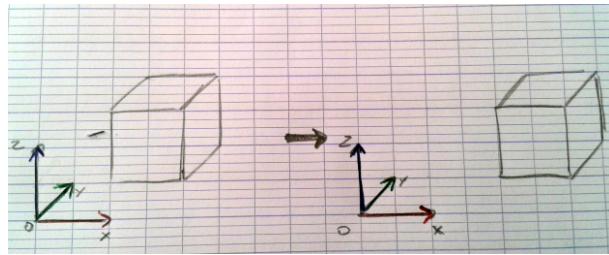


FIGURE 3 : translation

une direction (translation), de changer leur orientation (rotation), ou encore d'effectuer un changement d'échelle (agrandir ou rétrécir).

Ces transformations sont réalisées grâce à ce qu'on appelle une matrice, qui peut être vu comme un tableau de valeurs numériques (dans notre cas un tableau de 4 colonnes et 4 lignes, donc 16 valeurs). La transformation s'effectue par un calcul mathématiques entre le sommet et la matrice.

La matrice responsable des transformations appliquée aux objets virtuelle et qui nous permet ainsi de situer nos objets par rapport au repère 3D est appelé matrice de modèle mais ce n'est pas la seule à intervenir.

En effet dans notre monde virtuelle nous avons aussi ce qu'on appelle une caméra qui va correspondre au point de vue de l'utilisateur. Elle est représenté par trois vecteurs caractéristiques : le vecteur "eye" qui indique la position de la caméra dans le monde virtuel, le vecteur "target" qui indique la direction à laquelle on regarde le vecteur "up" qui indique où se situe le haut. Ainsi si l'on souhaite voir un objet sous différents angles ou se déplacer dans la scène c'est cette camera qu'on bougera, la matrice de transformation associé est appelé la matrice de vue.

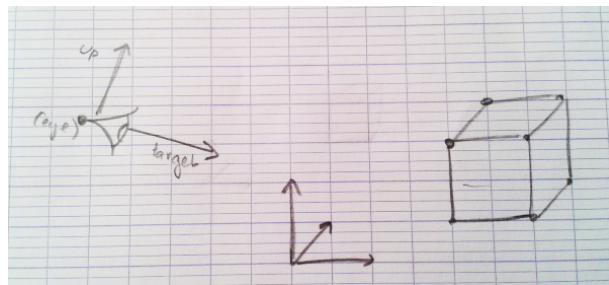


FIGURE 4 : Représentation de la camera dans la scène virtuelle

Dans la version d'OpenGL utilisé dans LC3D (version 2) on utilise qu'une seule matrice, la matrice ModelView qui est la combinaison de la matrice du Modèle et de la matrice de

vue.

A cela s'ajoute encore une autre matrice, la matrice de projection, qui elle sera responsable de notre champ de vision et de la perspective de la scène visualisé. Cela se manifeste entre autre par la capacité à distinguer un objet proche d'un objet éloigné, en appliquant une transformation qui comme notre perception du monde réel, nous fera apparaître les objets proche plus gros et les objets éloigné plus petit.

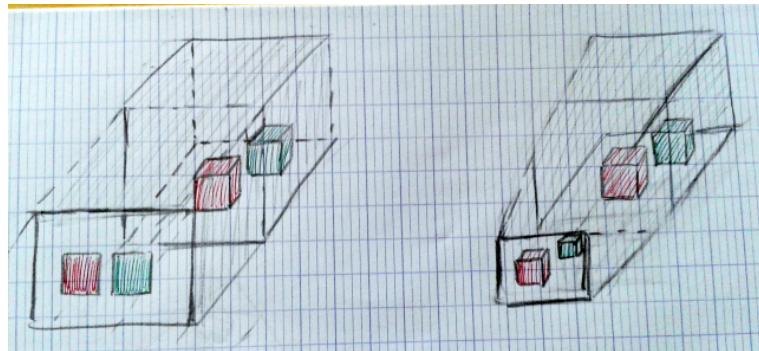


FIGURE 5 : Illustration du résultat obtenu avec une matrice de projection : le champ de vision de la caméra (frustum) est représenté par la zone grisée, la première est une projection orthogonale qui nous représente les deux cube à la même taille, la seconde est une projection en perspective qui nous fait apparaître le cube vert éloigné plus petit, ce qui correspond plus à la réalité

Maintenant dans la cadre de la réalité augmenté, pour faire la superposition avec une image réel, nous utilisons l'image en tant qu'image de fond. Ne ne considérons pas cette image comme un élément virtuelle, c'est-à-dire que quelque soit la valeur de notre matrice ModelView ou de notre matrice Projection, la visualisation de l'image reste inchangé.

On peut voir en figure 6 ce que l'on obtient désormais dans notre scène virtuelle.

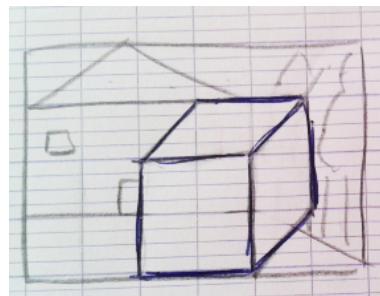


FIGURE 6 : Superposition d'une scène virtuelle sur une image réelle sans traitement préalable

Comme nous pouvons le constaté notre scène virtuelle superposé n'est pas du tout cohérente avec la scène réelle. Cependant comme vue plus haut, en modifiant la position de la caméra (matrice ModelView) et la matrice de Projection, nous pouvons modifier la visualisation de notre scène virtuelle et obtenir un résultat adéquat (Figure 7).

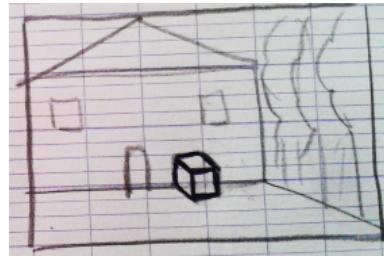


FIGURE 7 : Superposition d'une scène virtuelle après détermination de la matrice ModelView et la matrice Projection

Assurer cette cohésion entre la scène réelle et la scène virtuelle fait partie des grandes problématiques de la réalité augmentée. Trouver les matrices ModelView et Projection adéquatement sont réalisé dans LC3D grâce à un processus appelé "calibration".

la calibration de l'appareil chargé de capturé l'image réelle (appareil photo, camera...), consiste à récupérer des paramètre spécifique de la caméra (paramètre intrinsèque et extrinsèque). Ces paramètre nous permettront de retrouver les transformations que subit un point dans l'espace (point 3D) afin d'être projeté sur l'image résultat en 2D (pixel) . Autrement dit, cela revient à calculer ce qu'on appelle la "pose" de la caméra, c'est-à-dire la position de la caméra dans l'espace qui nous permettra pour un point 3D donné de le visualiser sur le pixel en 2D correspondant sur l'image réelle, c'est de là qu'on en déduit notre matrice de Projection et de ModelView.

Dans LC3D les points 3D , au nombre de 4 pour former un quadrilatère, sont défini à partir d'un objet de référence sur l'image réelle, appelé "la mire". La mire peut être de deux type, soit un objet posé sur le sol, soit une fenêtre. C'est également à cette étape qu'on en profite pour détecter le sol, en spécifiant que la coordonnée Z de nos points 3D est égale à 0. La longueur entre deux sommet doit également être spécifier, ce qui nous permet d'avoir le respect des mesures. Les pixels correspondant sont quand à eu, sélectionner par l'utilisateur par clique sur l'image réelle. Après cela nous pouvons créer notre scène virtuelle (Annexe création d'un projet avec LogyConcept3D Pool).

En ce qui concerne les occlusions, elle ne sont pas déduites automatiquement et doivent être masqué à la main grâce à une fonctionnalité du logiciel. Les ombres sont quand à elle affiché automatiquement en fonction de la position du soleil.

Même si nous obtenons un résultats correcte, le processus de calibration utilisé soulève

quelques contraintes et peut être amélioré, on aboutit ainsi au sujet de mon stage.

5 Présentation du sujet

Mon stage est surtout lié à la suite de logiciel LC3D, notamment celui de la piscine que j'avais à disposition.

Plusieurs problématiques interviennent vis-à-vis de l'implémentation existantes. En particulier, la "contrainte" la plus importante et que les utilisateurs concernés (piscinier, voiliste) n'ont aucune connaissance préalables en informatiques, et n'ont pour beaucoup jamais manipuler de logiciel. Cela nécessite que le logiciel soit simple et intuitif. Les principaux inconvénients liés à cela sont les suivants :

- Plus il y a d'action à réaliser, plus l'utilisateur risque de trouver cela compliqué :
 - L'utilisateur doit sélectionner quatre pixels dans un ordre correcte pour l'étape de calibration
 - Il doit faire le masquage à la main s'il veut gérer les occlusion
 - La prise de mesures réelle, nécessaire à la calibration, n'est pas toujours évidente.
 - Si la mire utilisée est celle vendue par l'entreprise, alors les dimensions sont connues, si non il faut que le client les prennent lui-même. En particulier pour la mire "fenêtre" il doit aussi spécifier la distance entre le sol et la mire.
- Niveau technique, le plan relatif au sol généré, peut présenter quelques incohérences en fonction de la nature du terrain (Figure 8)

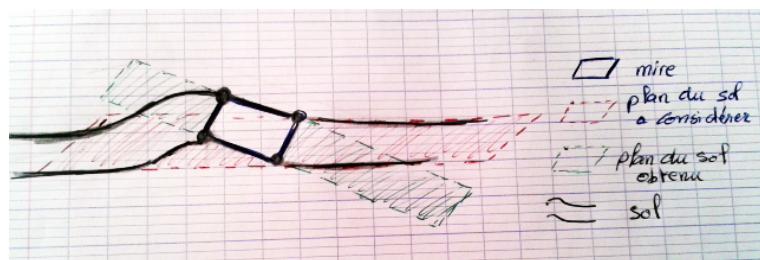


FIGURE 8 : Illustration du problème de détection du sol pouvant survenir à cause du terrain

L'objectif principal de mon stage consiste alors à simplifier et améliorer l'étape de calibration réalisé afin de permettre de corriger ou limiter l'impact des contraintes citées.

- En apportant des **modification** au code déjà existant
- En mettant en place de **nouvelle** méthodes de calibration

Mon sujet n'est pas alors constitué d'un projet défini à mettre en place et implémenter sur toute la durée du stage, mais de plusieurs pistes de **solution** à chercher et **exploiter**. L'intitulé de mon stage fait en fait partie d'une **des piste envisagée** qui semblait être dans un premier temps une solution idéale. J'expliquerai tout cela en détail dans la partie suivante.

6 Travail Effectué

6.1 Cadre et méthode de Travail

Je **travail** en autonomie au sein de l'entreprise, mais suis encadré par mon **tuteurs** et un autre ingénieur d'études et de **développement**. Ils ont **tout** deux **participé** au développement du logiciel LC3D et **travail** dans la même salle que moi, **il** sont ainsi à ma disposition pour toute **questions** ou aide **éventuelles**. Le mode de travail généralement appliquée au cours de mon stage peut être décomposé de la façon suivante :

- Phase 1 : Réunion avec mon tuteur et l'autre **developpeur** pour fixer les objectifs attendus
- Phase 2 : Prise d'information sur les technologies à utiliser, de ce qui se fait déjà et qui semble être intéressant à exploiter,
- Phase 3 : Développement de petites applications pour me permettre de prendre en main les technologies utilisées,
- Phase 4 : Développement d'une application répondant aux objectifs fixés.

Chaque phase étant **encadré** de petites réunions où les objectifs sont mis à jour en fonction de mes résultats.

6.2 Etat de l'art des technologies à **utilisées**

Avant toute chose il m'a été demandé de faire un état de l'art des technologies à utiliser en se posant les contraintes suivantes :

- LC3D est codé en C# et ne fonctionne que sur Windows
- La prise de vue concerne un environnement extérieur.

On aimerait également avoir une notion des distances sans avoir à les mesurer préalablement. Cela permettrait d'une part de faciliter le processus de calibration, et d'autre part à gérer les occlusions. En effet, si on connaît la distance par rapport à la caméra de chacun des éléments constituant la scène réelle, alors on peut facilement déduire si l'objet **virtuelle** est masqué par un objet plus proche.

Ce travail a été réalisé au cours de la première semaine de mon stage.

Parmis les **technologie** et appareil notable pour faire la de la réalité augmentée, on peut distinguer :

- Les technologies avec marqueurs :

Utilisent un "marqueur", c'est-à-dire une image caractéristique pouvant facilement être détecté grâce à des algorithme efficaces. La principale différence avec le système de mire réside dans le fait que l'utilisateur n'a pas à spécifier à la main l'emplacement du marqueurs. On peut citer les caractéristiques suivantes : Le calcul de pose peut se faire automatiquement. La tailles des objets virtuelles sont en relation avec la taille du marqueurs, de ce fait, en spécifiant la taille réelle du marqueurs, l'échelle de mesure cohérente entre objet virtuel et réel est respecté. Il existe des bibliothèque gratuite et compatible avec l'implémentation existante de LC3D. Les occlusions ne sont pas traité automatiquement. Sensible aux jeux de lumière (luminosité, reflet, ombre...)

- Les capteurs :

Ce sont des dispositif présent dans la plupart des tablettes et smartphone qui récupère des données relatives à des informations diverse tel que la direction où se trouve le nord (boussole), l'inclinaison, la rotation de l'appareil etc. Ils sont surtout utiles pour déterminer la position et l'orientation de l'appareil au moment de la prise de vue. Ils peuvent également aidé à la détection du sol (partie capteurs)

Pour LC3D, cela implique l'utilisation d'une tablette fonctionnant sur Windows (le logiciel n'est pas compatible avec ios et android). Néanmoins, certains client du logiciel possède déjà un tel matériel, et ne sont pas contre de se procurer un si cela apporte une meilleure simplicité d'usage.

- Les caméras 3D :

Ce sont des cameras qui nous donne en plus une notion de profondeur (distance par rapport à la caméra) au moment de la prise de vue. Cette troisième donnée nous permettra ainsi aisément de déterminer des points 3D, de calculé la distance réelle entre ces points, et même

de géré des occlusions. Ce sont ces derniers dispositifs qui correspondait le plus aux attentes de base, c'est pour cela que le terme vision stéréoscopique (détailé partie suivante) et au centre de l'intitulé de mon stage.

J'ai donc axé principalement mes recherche sur ce type de caméra, en privilégiant les utilisations en extérieurs. Deux caméras sortait du lot, il s'agissait de la camera "RealSense" de Intel, et de la "Zed" camera de Stereolab.

Dans la figure 9 sont listé les principales caractéristiques de chacune des deux caméras :

| | Real Sens | Zed Camera |
|---|--------------------------------------|---|
| Résolution image couleur | 1920x1080 à 30 ips | 4416 x 1242 à 15 ips 3080 x 1080 à 30 ips 2560 x 720 à 60 ips 1344 x 376 à 100 ips |
| Résolution Image de profondeur | 640 x 480 à 60 ips | idem que résolution couleur |
| Champs de profondeur | 10 mètres | de 1 à 20 mètres |
| Connexion | USB 3.0 | USB 3.0 |
| Dimensions (l x h x p) | 130 x 20 x 7 mm | 175 x 30 x 33 mm |
| OS compatible | Windows 8.1 et plus | Windows 7 et plus ; Linux |
| Configuration requise | 4ème génération de processeur Intel® | Dual-core 2,3GHZ ou mieux 4 GB RAM ou plus Carte Graphique NVIDIA |
| Langages de programmation utilisé dans la SDK | C++, C#, Java, JavaScript | C++ |
| Prix | 99 \$ | 449 \$ |

FIGURE 9 : Tableau comparatif des principales caractéristiques des caméras realense et zed

6.3 Piste envisagé 1 : Utilisation des caméras 3D

Problèmes concernés :

- Nombre d'actions utilisateurs pour le processus de la calibration
- Occlusions

Objectifs visés :

- Effectuer l'étape de détection de la mire en moins de quatre clics (et dans l'idéal ne pas avoir de mire)
- Gérer les occlusions
- Pouvoir mesurer des distances réelles à partir de la photo

Intérêt :

- Si la capture d'image est réalisée avec un même appareil, les paramètres récupérés lors de la calibration seront les mêmes. L'utilisateur pourra ainsi s'abstenir d'effectuer l'étape post-calibration qui consiste à photographier un damier sous différents angles de vue.
- La notion de profondeur peut permettre de gérer les occlusions et de mesurer des distances

Principe de fonctionnement pour l'obtention de la profondeur : la vision stéréoscopique. :

Le principe de la stéréoscopie se base sur la vision humaine : avec l'écart de 6 cm en moyenne se trouvant entre nos deux yeux, en obtient deux images décalées représentatives de la même scène (une pour l'oeil droit, et une pour l'oeil gauche). Notre cerveau combine ensuite ces deux images pour nous retourner une image en trois dimensions grâce à la perception de la profondeur. Ce fonctionnement peut être expliqué comme suit (ref) : Si deux images d'une scène sont acquises depuis différents points de vue, alors la profondeur de la scène crée une disparité géométrique entre elles. Soit ε la fonction liée à la disparité et z à la profondeur, on obtient la relation suivante :

$$z = \frac{\varepsilon}{b/h}$$

où b est la distance entre les centres des caméras et h est la distance entre la scène et les caméras. Cela est illustré dans la figure 10

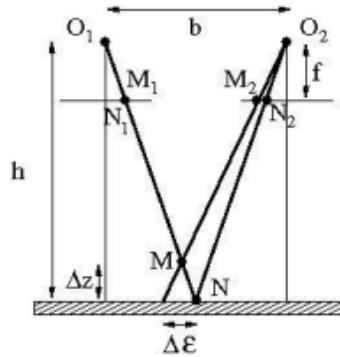


FIGURE 10 : Principe de la stéréoscopie (Julie Delon, 2007) : O_1 et O_2 représentent les centres des caméras. Les projections des points M et N dans la première image sont M_1 et N_1 , et M_2 et N_2 pour la seconde. Nous pouvons voir que la position de M_1 dans la première image n'est pas la même que celle de M_2 dans la seconde. Notons ΔM le décalage entre ces positions (respectivement ΔN). La différence des décalages $\Delta M - \Delta N$ est proportionnel à la différence de profondeur ΔZ (le coefficient de proportionnalité étant b/h)

6.3.1 Realsense



FIGURE 11 : Caméra RealSense R200 de Intel

Ce modèle de caméra est normalement conçu pour l'extérieur pouvant capter une profondeur allant jusqu'à dix mètres. Elle a le mérite d'être petite et légère et peut se fixer rapidement à tout type de matériel.

Concernant le calcul de la profondeur, elle fonctionne par projection infrarouge. Principe : Plusieurs rayons infrarouges sont envoyés, et ceux-ci se réfléchissent sur les éléments de l'environnement face à la caméra. Les rayons réfléchis sont visibles grâce à une caméra infrarouge.

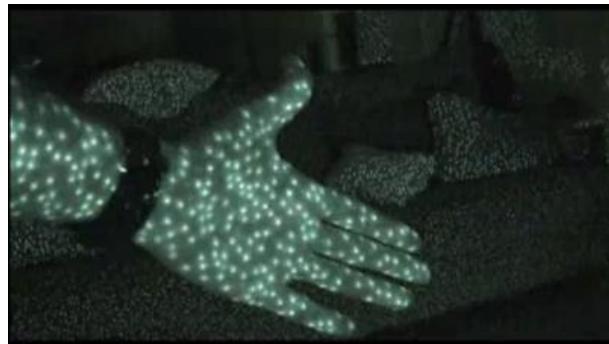


FIGURE 12 : Projection de points infrarouges via une kinect

Plus un élément est éloigné et plus la quantité de rayonnement infrarouge **réfléchie** sera faible et inversement. On retrouve également ce procédé au sein de la kinect de microsoft.

La particularité de la realsense est d'utiliser deux **caméra infrarouge**, c'est à partir de ces deux **camera** que la stéréoscopie est **mis** en place (Figure 13).



FIGURE 13 : Disposition des différents composants de la realsense

Afin de tester cette camera j'ai **exploiter** sa Software Development Kit, en français : Kit de développement (SDK) qui fournit un certain nombres d'applications **prête** à l'emploi (représentation 3D d'un modèle filmé, calcul de distance en sélectionnant deux **point** sur image, génération de la carte de profondeur)

On obtient des résultats assez **satisfaisant** en intérieur à condition que le profondeur soit bien détecté, comme l'illustre les figures 14 et 15

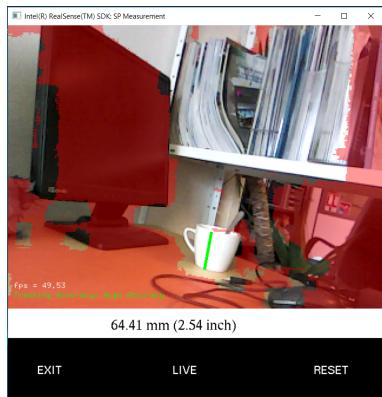


FIGURE 14 : Calcul des mesures avec la camera realsense, la mesure réelle est de 65 mm, l'espace rouge correspond à la zone où la profondeur n'a pas pu être obtenu.

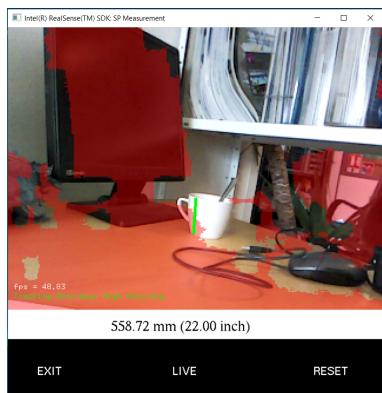


FIGURE 15 : Lorsque que la profondeur n'est pas détectée correctement, alors les résultats obtenu sont faussés.

Ce qu'on remarque le plus avec les figures 14 et 15 c'est que la profondeur est très mal détecté alors que nous sommes en interieur et donc dans de meilleures conditions (pas de lumière du jour, élément plus proche) . Les test effectué en exterieur n'ont fait que confirmé ces mauvais résultats, en effet je n'obtenais casiment pas cette donnée dans la plus part des cas...

Le système de projection infrarouge en est la cause principale, une trop forte luminosité empêche une bonne détection des points projeté, les surfaces noires, brillante et réfléchissante fausse également le résultat.

C'est une assez grandes contraintes car nous voulons un résultat pouvant fonctionné tout le temps et non dans certaines conditions. Il a donc été décidé d'abandonner l'idée d'utiliser

cette caméra. Nous nous sommes alors tourné vers la ZED caméra.

6.3.2 ZED



FIGURE 16 : ZED Camera de Stereolabs

En ce qui concerne la ZED camera, il n'y a pas de projection infrarouge et la stéréoscopie s'effectue avec deux images capturées séparément de 12cm.

On peut voir dans la figure fig un comparatif des résultats obtenus avec la zed caméra et la realsense

La profondeur est mieux détectée avec la ZED Caméra, mais celle-ci est toute fois moins précise. Il y a tout de même une grosse contrainte qui nous empêche d'avancer comme on le souhaite. Cette fois on se place du point de vu du client. Le problème est surtout matériel et financier, pour calculer la profondeur la zed camera a besoin de "cuda", une technologie associée aux cartes graphiques NVIDIA pour une plus grande rapidité de calcul. Il est bien sûr inconcevable de se balader avec un ordinateur portable à la main pour capturer la photo, malheureusement peu de tablette windows correspondent à ce profil. Les rares appareils compatibles avoisinent un prix autour des 400\$, ajouté au prix de 440\$ de la camera, on atteint les 840\$ à payer en plus du prix de base de LC3D (69\$ par mois, considéré déjà comme trop cher pour certains clients).

Il a été convenu donc de ne pas trop s'attarder sur cette technologie, un des intérêts de mon stage étant de pouvoir fournir à l'entreprise quelques chose qu'il pourront exploiter et mettre en oeuvre au service du client, dès l'année prochaine.

J'ai donc été dirigé vers un problème existant au sein de la calibration de LC3D qui peut être trouvé une solution grâce à l'utilisation des capteurs.

ajouter figure

6.4 Piste envisagée 2 : Utilisation des capteurs

Problème concerné : le plan relatif au sol généré, peut présenter quelques incohérences en fonction de la nature du terrain

6.4.1 Objectif

Le "mauvais" plan détecté est corrigé à la main dans LOGYConcept en essayant de terminer des valeurs entre la mire et le sol. Ces valeurs sont utilisées afin de faire une rotation autour de l'axe X et Y (figure 17). L'idée est alors automatisé ce processus en essayant déduire ces valeurs grâce aux capteurs.

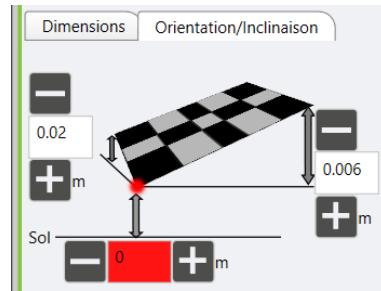


FIGURE 17 : Outils de correction de plan dans LC3D

6.4.2 Information sur les technologies à utiliser

On s'interessera plus particulièrement au capteurs accéléromètre, gyromètre et boussole.

- L'accéléromètre permet de savoir dans quelle direction l'appareil se déplace, il retourne une valeur d'accélération (vitesse d'un mouvement en fonction du temps) par rapport aux axes X, Y et Z.
- Le gyromètre détecte une accélération de rotation des axes X, Y et Z

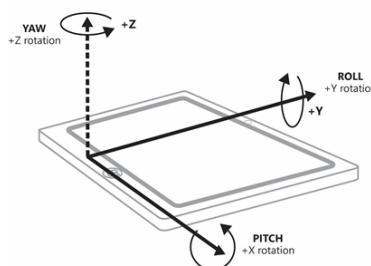


FIGURE 18 : Représentaion d'une tablette et son repère pris en considération par les capteurs d'accéléromètre et de gyromètre

- La boussole retourne la valeur de l'angle en degrés (de 0 à 360°) entre l'axe **horizontale** de l'appareil et le nord magnétique.

Ces capteurs sont **disponibles** dans la SDK de Windows 10 et peuvent être **utilisés** au sein d'une application **codée** via VisualStudio. On y trouve également un capteur d'orientation, **qui** une combinaison des trois **autres** capteurs cités plus haut et qui retourne une matrice de rotation relative à l'orientation de la tablette.

Afin de manipuler ces capteurs, j'avais à disposition un Microsoft Surface 3.

6.4.3 Prise en main des capteurs

Avant toute chose il me fallait me familiariser avec l'utilisation de ces capteurs, j'ai donc créé dans un premier temps des applications typiques de réalité virtuelle qui me **permettait** de modifier le point de vue utilisateur en fonction de l'orientation de la tablette. La réalité virtuelle se distingue de la réalité augmentée par une scène uniquement virtuelle sans présence de quelconques éléments réels.

J'ai conçu trois applications dont le principe est schématisé dans les figures 19 et 20.

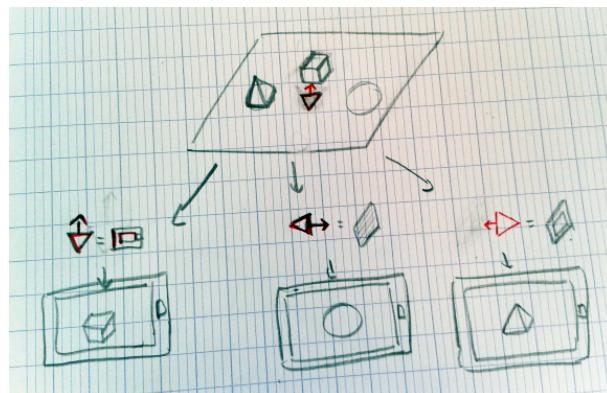


FIGURE 19 : Principe de l'application 1 de prise en main

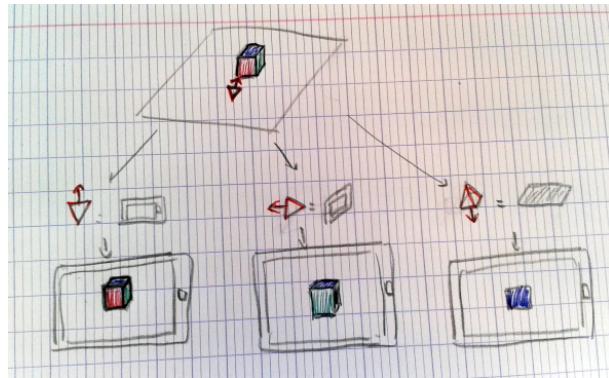


FIGURE 20 : Principe de l'application 2 et 3 de prise en main

J'ai d'abord commencé par modéliser une scène en OpenGL. J'ai utilisé le même langage de programmation que l'entreprise, à savoir le C#. Il s'agit d'un langage nouveau pour moi, mais étant très proche du langage JAVA dont j'avais déjà connaissance de par ma formation en licence et via l'UE "Approche Objet" de la première année du master, je n'ai pas eu de grosses difficultés à l'utiliser. OpenGL étant destiné au langage C++, je me suis servi d'OpenTK qui est ce qu'on appelle un "Wrapper". Il va en quelque sorte englober les fonctionnalités d'OpenGL et les adapter de façon à être compatible avec le C#. Pour la modélisation de la scène virtuelle, mon tuteur m'avait fourni un extrait de code pour me simplifier la tâche. Connaissant déjà les bases d'OpenGL via différents enseignements de la spécialité image, son, vidéo du master, l'extrait donné ma surtout permis de comprendre le fonctionnement du wrapper OpenTK au sein d'un projet "Windows Form" sur Visual Studio. Les projets Window Form nous offre une interface graphique prêt à l'emploi, que l'on peut facilement modifier en glissant-déposant les différents éléments (boutons, labels, fenêtre de vue d'OpenTK). Cette extrait de code m'a également permis de programmer avec la version 2 d'OpenGL, la version vu à l'université étant la 3.3. Dans mon cas la principale différence était la non utilisation des "shaders", qui sont des programmes informatiques sollicitant la carte graphiques afin d'effectuer le rendu 3D. En particulier c'est à eux que sont transmis les informations relative au Matrice de Modèle, Vue et Projection. Il faut savoir que plusieurs entreprises dont Logyline sont resté à la version 2 d'OpenGL car changer de version impliquerai de grosse modifications dans le codes déjà existants.

La première application créée consiste à modifier la direction à laquelle on regarde, il s'agit d'un procédé similaire à ce qu'il se passe lorsque l'on bouge la tête en visualisant une scène via un casque de réalité virtuelle. Pour ce faire je me suis servi du capteur d'orientation qui me retourne une matrice de rotation. Pour aboutir à l'objectif désiré, j'ai appliqué cette matrice à la position de la caméra.

Difficultés rencontrées

- Le repère 3D à partir duquel est créée la matrice de rotation renvoyé par le capteur n'est pas le même que celui-ci représenté dans la scène virtuelle créée. Il fallait dans un premier temps faire correspondre ces deux repères (figure 21)



FIGURE 21 : Transformations appliquées pour passer du repère de la tablette au repère de la scène virtuelle

- La rotation de la caméra implique une translation comme illustré à la figure 22, j'étais déjà tombé sur ce cas de figure lors d'un TD au sein de l'UE Réalité virtuelle où il fallait effectué une rotation de la vue utilisateur en bougeant la souris. J'ai pu ainsi réglé ce soucis assez rapidement et ai obtenu de meilleures résultats qu'en TD.



FIGURE 22 : Rotation de la direction de la vue

- Afin d'effectuer les calcul entre matrice et vecteur, j'ai dans un premier temps chercher une bibliothèques de calcul matriciel en C# mais n'avais rien trouvé qui me convenais. J'ai alors pris l'initiative de créer moi même cette bibliothèque en implémentant les calculs de base et en créant notamment les type de donnée Vecteur et Matrice. Au final, cela à surtout été une perte de temps car OpenTk implémentait ces fonctionnalités. Cependant cela ma également fait prendre conscience d'une autre erreur, au moment de la récupération des données de la matrice de rotation. Le capteur d'orientation ne nous renvoie pas une matrice toute faite mais chacune des valeurs de chacun des éléments dont celle-ci est constitué. Je crénais donc une matrice en initialisant les éléments ligne par ligne. En utilisant ma propre bibliothèque et mes propres type de donnée j'obtenai le résultat voulu, mais ce n'était pas le cas en utilisant les fonctionnalité fournit pas OpenTK. En effet OpenTK initialise une matrice colonne par colonne, en sachant cela j'ai pu évité d'autres erreurs éventuelles pour les prochaine application.

En ce qui concerne les deux autres applications, il s'agissait du même principe sauf que je déplace également la caméra. Pour l'application 2 il me suffisait de ne pas appliquer la translation évoqué plus haut, pour l'application j'ai utilisé n'ont pas le capteurs d'orientation mais l'angle retourné par la boussole. Ainsi je n'ai pas récupéré de matrice, mais simplement effectué une rotation autour de l'axe Z avec l'angle obtenue.

6.4.4 Création d'une application répondant à l'objectif fixé

Principe : Pour rappel l'étape de calibration nous positionne notre camera virtuelle de façon à ce que la scène virtuelle s'intègre de manière logique dans la scène réelle. Lorsque le plan est mal détecté, cela est corrigé en effectuant une rotation. Cette rotation revient en fait à changé l'orientation de la caméra virtuelle. On part avec comme hypothèse qu'en attribuant à la caméra virtuelle la même orientation qu'avait la caméra au moment de la prise de vue, cela permettra de corriger le plan.

Dans un premier temps il me fallait concevoir une application qui me permettrait de récupéré l'information des capteurs. Il m'a été conseillé d'enregistrer ses informations au sein d'un fichier XML afin de facilité la récupération des données. J'ai alors conçu une application, toujours en C# avec la bibliothèque Emgu, qui est un wrapper d'OpenCV, une bibliothèque graphique spécialisé dans le traitement d'images. Emgu me servira à utiliser la webcam de la tablette. L'image filmé est ensuite enregistré par un clique sur un bouton, au même moment les informations des capteurs sont enregistrer dans un fichier XML.

Dans un deuxième temps, pour exploiter ses données j'ai conçu une deuxième application reproduisant l'étape de calibration de LC3D. En guise de mire, j'ai utilisé une feuille de 21cm x 14,8 cm (la mire de Logyline est de 1m40 x 1m40). N'ayant pas accès aux codes sources du logiciel, mon tuteur m'as transmis le bout de code correspondant. Cela m'a pris un certains temps, pour deux principale raisons : La première était que le code donné était dépendant d'autres bibliothèques propres à LC3D. Ces bibliothèques concernant surtout la mise en place de calcul basique, et la création de type Vecteur et Matrice, j'étais en mesure de les recoder avec le stricte minimum. On ne se sert volontairement pas des types de données propre à OpenTK, afin de pouvoir reprendre le procédé de calibration dans d'autre logiciel ne fonctionnant pas forcément avec cette bibliothèque. La deuxième raison est que la matrice de ModelView positionne la caméra en bas de la scène virtuelle. Je ne me suis rendu compte de cela qu'après avoir comparé mon implémentation avec celle de LOGYConcept3D en présence de mon tuteur. Une fois cette étape réalisé, afin d'avoir une meilleure visualisation du travail fourni, j'ai mis en place trois de type de cameras, représentative de trois vue différentes :

- "CalibratedCam", la camera où s'est effectué la calibration
- "SurfaceCam", la caméra représentative de l'orientation de la Microsoft Surface

- "SpectatorCam", nous permettant d'observer les deux autres caméras.

L'interface graphique relative à mon implémentation est représenté par la figure 23



FIGURE 23 : Interface graphique de l'application créée

Au chargement de l'image de fond, capturé par la première application, on récupère également les données de la matrice de rotation, récupérée par le capteur d'orientation, contenu dans le fichier XML associé. On applique cette matrice de rotation à "SurfaceCam", le visuel obtenu est illustré dans la figure 24

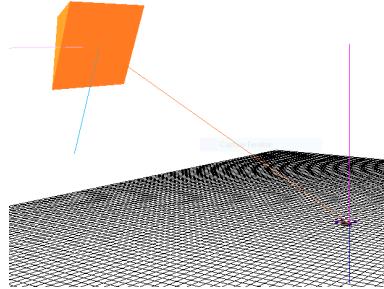


FIGURE 24 : Orientation de la tablette au moment de la photo

Nous n'avons connaissance que de l'orientation de la caméra, en particulier sa position par rapport au sol n'est pas connue. Dans la figure 24 il s'agit d'une position mise par défaut. La figure 25 nous montre ce que nous obtenons après la calibration.

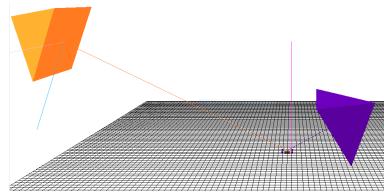


FIGURE 25 : Orientation de la caméra après calibration

L'idée maintenant est de positionner "SurfaceCam" à la même position que "Calibrated-Cam", pour ce faire on procède comme présenté dans la figure 26,

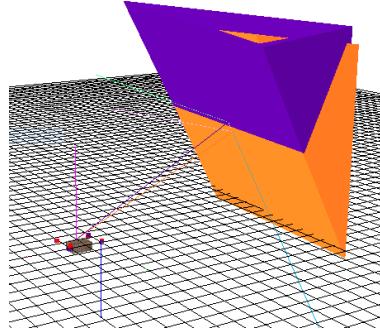


FIGURE 26 : Superposition des deux caméras

On remarque dans la vue de "SurfaceCam" que le plan paraît déjà plus **redresser**. Par la suite je récupère la plus grande distance entre **le points** 3D délimitant ma mire dans le plan de "CalibratedCam" et dans le plan de "SurfaceCam". Je rentre ces données dans **le partie réservé** à la correction du plan dans LC3D ce qui redresse effectivement le plan.

Malheureusement, tout cela ne **fonctionnement** pas dans tous les cas... Dans certaines situations, même si les deux caméras sont **aligné**, le plan est mal détecté par "SurfaceCam", du coup aucune correction ne peut être **effectué**. Un plan correcte peut tout de même être obtenu, mais en changeant la position de "SurfaceCam" à la main, ce qui n'a pas d'intérêt. Je me suis attardé à **trouvé** une solution mais sans succès, il m'a donc été **conseiller** de passer à autre chose, mon tuteur ayant en tête une autre piste pour améliorer la calibration existante

6.5 Piste **envisagé** 3 : Utilisation d'Aruco

Problèmes concernés :

- Nombre d'actions utilisateurs pour le processus de la calibration

Objectifs visés :

- Effectuer l'étape de détection de la mire en moins de quatres clics (et dans l'idéal sans clic)
- Amélioration du logiciel en permettant une nouvelle fonctionnalité : travailler avec une vidéo filmée en temps **réels** plutôt qu'une image fixe

6.5.1 Information sur les technologies à utiliser

Aruco est une bibliothèque basé sur OpenCV pour faire de la réalité augmentée à l'aide de marqueurs. Les marqueurs sont de forme carré et possèdent un motif particulier qui correspond à une codification. Des algorithmes de reconnaissance vont permettre de reconnaître cette codification et ainsi détecté l'ensemble du marqueur. Une fois le marqueurs détecté les sommets relatifs à ses quatre coins sont récupérés automatiquement afin d'effectuer le calcul de pose qui nous permettra alors de positionner un objet virtuel.



FIGURE 27 : Exemple de marqueur utilisé dans ArUco

Pour générer une scène virtuelle grâce à aruco il nous faut alors garder en visuelle le marqueur. Dans un flux vidéo, ce mécanisme de détection aura pour effet de continuellement mettre à jour les données concernant la position du marqueur et donc de chacun des objets de notre scène. On parle alors de "suivi de marqueur" où plus exactement de "*tracking*". Le tracking est une étape importante au sein dans la réalité augmentée avec vidéo, il permet de conservé une cohérence de la scène virtuelle lorsque l'utilisateur se déplace.

Des essais avec ArUco avaient déjà été effectués par l'entreprise au cours de l'année 2010, mais les résultats n'était pas assez concluant pour pouvoir être implémenter au sein des logiciel existants. Le problème venait du fait que le marqueur n'était détecté uniquement qu'à très courte portée.

Cependant la bibliothèque n'a pas cessé d'évoluer, la dernière version à ce jour étant daté du 29 juillet 2016. Il m'a donc été demandé de renouveler l'expérience.

6.5.2 Prise en main d'ArUco

ArUco propose des exemples simples pouvant être compilé et testé. De même j'avais déjà eu l'occasion d'utiliser cette bibliothèque au cours de l'UE : Réalité virtuelle du master. Cette étape de prise en main c'est donc déroulé assez rapidement et je ne me suis pas attardé dessus. Néanmoins il important de mentionné que jusqu'à présent j'avais manipuler cette bibliothèque en C++. Il n'y avait donc pas de difficultés apparente, Aruco dépendant d'OpenCV et OpenCV disponible en C++, je disposait de toute les ressources nécessaires.

6.5.3 Création d'une application répondant à l'objectif fixé

L'application à créer doit être en mesure d'effectuer les fonctionnalités de base d'Aruco (déttection de marqueur, extraction de la matrice ModelView). Mais pour pouvoir être utilisée par la suite dans LC3D, il faut qu'elle soit codée en C#. C'est de là que vient la principale difficulté rencontrée. Dans un premier temps j'ai essayé de programmer une bibliothèque externe, qui va consisté à utiliser les fonctionnalités d'aruco en C++ et les retourner dans un format lisible par le langage C#. C'est cette procédure que mon tuteur avait appliquée en 2010. J'ai alors créé une fonction en C++ qui effectue le code détection de marqueurs et d'extraction de la matrice modelView. Cette fonction me retourne le nombre de marqueur détecté, la matrice de projection, et la matrice de modelView. Pour obtenir la matrice de projection et modelview, il me faut spécifier la taille du marqueur ainsi que l'emplacement du fichier ".yml", un format dérivé du ".xml", qui contient les paramètres de calibration de la caméra. Il n'y pas de soucis de compatibilité pour la transmission de valeur numérique entre C# et C++ (nombre de marqueurs détecté, taille du marqueur, valeurs contenues dans les matrices de projection et modelview), mais il y en a un pour le type "string" correspondant aux chaînes de caractères (qu'on utilise pour spécifier l'emplacement du fichier ".yml"). Cela s'explique car le type "string" en C++ correspond en réalité à une classe possédant des attribut et des méthodes, alors nous ne voulons transmettre que des données. Le problème se règle en utilisant "*char" en C++ qui correspond au type "string" en C#. Malgré ce premier problème résolu, mon programme ne fonctionnait pas et me générera des erreurs vis-a-vis de ma bibliothèque créée. J'ai essayé de comprendre ces erreurs tant bien que mal, mais sur le coup je n'ai pas réussi à les résoudre. En cherchant des idées de solution sur le net, j'ai réalisé qu'il existait un "wrapper" d'aruco en CSharp nommé "Aruco.Net".

J'ai alors décidé de mettre de côté la dll externe, et d'intégrer le wrapper. J'ai alors pu obtenir un résultat fonctionnel sans difficulté. J'avais maintenant un marqueur détectable sur lequel je pouvais positionner un objet 3D. Maintenant pour renforcer cette détection, l'idée est utilisée un ensemble de marqueurs, de façon à ce que si l'un est partiellement caché, notre objet reste correctement positionné dans la mesure où d'autres marqueurs sont entièrement visibles. Cette fonctionnalité est disponible sur Aruco, elle considère une maquette de capteurs appelé "board". Si on se place dans le cadre de LC3D, les marqueurs seront positionnés sur une bâche qui sera superposé sur la mire de 1m40 par 1m40 actuellement commercialisé. Sachant que plus un marqueur est grand et plus il sera visible de loin, la planche de marqueurs à considérer ne doit pas contenir un trop gros nombre de capteurs. Quatre était le nombre idéal à considérer, chaque marqueur occupant un espace égale et exploitant toute la surface disponible sur la mire carré.

Le problème qui se pose désormais, c'est qu'"Aruco.Net" ne fournit pas la fonctionnalité permettant de traiter un "board". Je ne pouvais pas non plus l'implémenter à la main car je disposais pas des droits nécessaires pour modifier le wrapper...

6.6 Bonus : Gestion des ~~occlusion~~ avec la ZED caméra

7 Bilan et perspectives

Glossaire

bibliothèque (informatique): ensemble de ~~fonction~~ informatiques ~~prête~~ à l'emploi , ~~destiné~~ à être utilisé par d'autre programmes. 6, 26

calibration : Procédé permettant d'obtenir les paramètres ~~intresequel~~ et ~~extrinseque~~ d'une caméra ou appareil photo.. 9

extrinseque : Paramètre ~~externes~~ d'une caméra, il ~~comprends~~ les informations relatives aux ~~transformation~~ (rotation et translation) ~~nécessaire~~ au calcul de la pose.. 28

intresequel : ~~Paramètres internes~~ d'une camera, il comprend notamment la distance focale et le ~~coordonnée~~ du centre optique de la caméra, ~~nécessaire~~ pour obtenir la matrice de projection.. 28

Kit de développement : ensemble d'outils destinés aux développeurs, permettant de faciliter le développement d'un logiciel sur une plateforme donnée. 16, 28

nord magnétique : Correspond à l'un des points par lesquels passe l'axe du champ magnétique terrestre, il se distingue du nord géométrique qui correspond au pôle nord, l'un des points par lesquels passe l'axe de rotation de la Terre. 20

temps réel . 5

XML : langage informatique permettant de décrire des données à l'aide ~~de balise~~ et de règles que l'on peut personnaliser . 23, 24

Acronyme

LC3D LogyConcept3D. 5–7, 9–12, 18, 23, 25, 27

RA réalité augmentée. 4, 5

SDK Software Development Kit, en français : Kit de développement. 16, 20

Références

- [1] F. Kaghat, F. Sailhan, and P. Cubaud. Application de la réalité augmentée sonore pour les visites de musées par les malvoyants. In *Handicap 2012, 7ème congrès sur les aides techniques pour les personnes handicapées.*, page 50, Paris, France, July 2012.
- [2] Magic Leap Inc. <https://www.magicleap.com>, 2016. [en ligne ; consulté le 20 août 2016].
- [3] Microsoft. Hololens. <https://www.microsoft.com/microsoft-hololens/en-us>, 2016. [en ligne ; consulté le 20 août 2016].
- [4] NianticLabs. Pokémon go. <http://pokemongo.nianticlabs.com>, 2016. [en ligne ; consulté le 20 août 2016].