VRAI GROUP

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

FUSION D'INFORMATION TRIDIMENSIONNELLE SUR FLUX VIDÉO

Département de Microtechnique
Rapport projet de semestre d'hiver 2001-2002

Professeur : Reymond Clavel, ISR
Assistant : Nicolas Chauvin, VRAI Group / ISR

EPFL / VRAI Group Pierre-Olivier Latour

TABLE DES MATIERES

DONNEE DU PROJET

Introduction

Dans le cadre d'un projet de recherche entre le groupe VRAI et l'entreprise 2C3Dmedical, de nouvelles interfaces de visualisation graphique pour le domaine médical sont évaluées. Le but de ce projet est de pouvoir visualiser des objets virtuels tridimensionnels au sein d'un flux vidéo représentant une scène réelle (Augmented Reality Overlays).

Cahier des charges / travail demandé

Afin d'obtenir une bonne immersion, un casque de réalité virtuelle sera utilisé comme moyen d'affichage. Le projet comporte les phases suivantes :

- Fixation mécanique d'un tracker 3D et d'une webcam FireWire sur le casque de réalité virtuelle
- Tracking simplifié et estimation des mouvements de caméra (camera motion) pour la calibration initiale
- Affichage en temps réel d'objet 3D sur un flux vidéo live
- Tracking en orientation (HPR) des mouvements de la tête à l'aide d'un tracker InterSense
- Fusion de l'information de "camera motion" avec celle du tracker InterSense

RESUME DU PROJET

La réalité augmentée consiste à superposer des images virtuelles générées par ordinateur sur le monde réel, afin de fournir des informations supplémentaires à l'observateur.

Dans le prototype de système de réalité augmentée réalisé pour ce projet, nous filmons l'environnement réel à l'aide d'une caméra vidéo connectée à un ordinateur. Un logiciel spécifiquement développé pour l'occasion calcule les images virtuelles et les superpose sur les images vidéos, avant de les afficher dans des lunettes 3D. La synchronisation entre les mondes virtuel et réel s'effectue grâce à un cadre de boules de couleurs placé dans l'environnement et à un capteur d'orientation 3 DDL. La caméra vidéo et le capteur sont montés sur les lunettes 3D.

Les problèmes traités dans ce projet sont :

- choix du matériel à utiliser (caméra, capteur, lunettes 3D…),
- caractérisation de la caméra (focale, distorsion optique, dynamique des couleurs…),
- capture d'images vidéo depuis une caméra FireWire et décompression des données vidéos YUV,
- tracking de boules de couleurs,
- calcul de la position dans l'espace d'un cadre à partir de l'image de sa projection dans le plan focal de la caméra,
- lecture et décodage des données retournées par le capteur InterTrax 2 USB,
- import d'objet 3D au format OBJ de d'images au format TGA,
- rendu d'objet 3D et superposition sur les images vidéos avec OpenGL,
- fusion des informations du capteur et du cadre de boules de couleurs(*).

Deux applications sont testées avec le prototype :

- une dans le domaine médical qui consiste à afficher des organes à l'intérieur du patient,
- une dans le domaine de la maintenance où l'on affiche des câbles et des tuyaux à l'intérieur d'un mur.

(*) Des problèmes survenus à la dernière minute avec le capteur d'orientation InterTrax 2 ont empêché la réalisation de ce dernier point.

INTRODUCTION A LA REALITE AUGMENTEE

Nous aborderons dans cette introduction le principe de la réalité augmentée, son utilité et ses possibilités. Puis nous présenterons le prototype construit pour le projet.

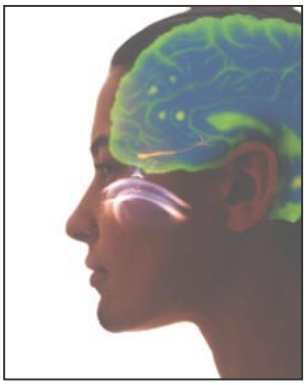
Présentation du concept

L'un des problèmes rencontré en réalité virtuelle est l'impossibilité de reproduire de façon réaliste l'environnement, et notamment la quantité phénoménale d'informations qu'il contient. Par exemple, si l'on souhaite reproduire virtuellement une pièce et son mobilier en 3 dimensions, on reproduira sans problème les tables, chaises, lampes… mais il est impossible de reproduire chaque aspérité des matériaux, les tâches sur le sol, les ombres aussi précisément… Si l'on souhaite reproduire une scène extérieure (une forêt par exemple), l'écart entre le réel et le virtuel sera encore plus grand.

La Réalité Augmentée est une branche de la recherche dans le domaine de la Réalité Virtuelle qui apporte une solution acceptable à ce problème. L'idée est de créer une image composite pour l'observateur comprenant, d'une part, *une vue du monde extérieur réel*, et d'autre part, *des images virtuelles superposées*. Ces dernières apportent des informations supplémentaires à l'observateur lui permettant de mieux percevoir son environnement. Le but ultime de la réalité augmentée est que l'observateur ne puisse distinguer les éléments virtuels des éléments réels.

Il est important de bien distinguer "Réalité Virtuelle" et "Réalité Augmenté". Dans le premier cas, l'utilisateur est complètement *immersé* dans un monde artificiel et ses sens perdent tout contact avec la réalité. Cela comprend bien sûr sa vision (au moyen d'un casque généralement), voir son sens du toucher et son audition. Dans le cas de la réalité augmentée, on "complémente" le monde réel, ce qui nécessite que l'utilisateur ait toujours l'impression d'être présent dans le monde physique. Cela suppose un mécanisme pour combiner et synchroniser les images virtuelles et réelles qui n'est pas requis dans le cas de la réalité virtuelle.

Les casques de pilotes de chasse par exemple (image ci-contre) utilisent une application élémentaire du principe de réalité augmentée. À travers le viseur transparent, le pilote voit toujours le monde réel mais avec des informations superposées provenant des instruments de bord, d'une caméra infrarouge, de son copilote… Ces informations sont projetées sur le viseur depuis le bras qui le tient.



Il existe des applications nettement plus complexes de la réalité augmentée. Par exemple, dans le domaine médical, on peut imaginer la superposition de données tirées d'un scanner préopératoire sur la tête du patient (image ci-contre). On afficherait également toute autre information utile provenant des appareils électroniques comme le pouls, la pression sanguine…
Offrir une telle possibilité au chirurgien lors de l'opération lui permettrait d'être beaucoup plus efficace.

Principe de réalisation

L'élément principal de la réalité augmentée est donc la superposition d'images virtuelles sur le monde réel. Le résultat est visible pour l'utilisateur grâce à un "casque" comprenant un système d'affichage. Il en existe deux types :

1. Ceux qui projettent les images virtuelles sur une vitre spéciale à travers laquelle on voit le monde réel. Les casques de ce type sont extrêmement coûteux et sont réservés quasi-exclusivement aux applications militaires ou médicales.
2. Ceux qui embarquent plus simplement un ou des écrans miniatures LCD opaques. Il est alors nécessaire de leur adjoindre une caméra vidéo qui filme ce que verrait l'utilisateur s'il ne portait pas le casque. Les images virtuelles sont fusionnées avec les images de la caméra puis projetées sur les écrans LCD.

Pour que le résultat soit acceptable par l'utilisateur, il est indispensable que :

- les images virtuelles soient "synchronisées" correctement sur le monde réel et que cette synchronisation perdure lorsque l'utilisateur regarde ailleurs ou se déplace.
- la latence du système soit la plus faible possible c.à.d. que, lorsque le champ de vision de l'utilisateur change, le retard dans l'affichage soit minimal pour éviter le mal de cœur.

Implémentation dans le cadre du projet

Au vu des derniers progrès dans le matériel de réalité virtuelle, il nous a paru réaliste de chercher à réaliser un prototype "clé en main" de réalité augmentée, ce qui implique :

- Utilisation de matériel "grand public" pour un coût réduit,
- Bonne qualité visuelle,
- Simplicité de mise en œuvre et portabilité.

En respectant toutefois ces deux contraintes essentielles :

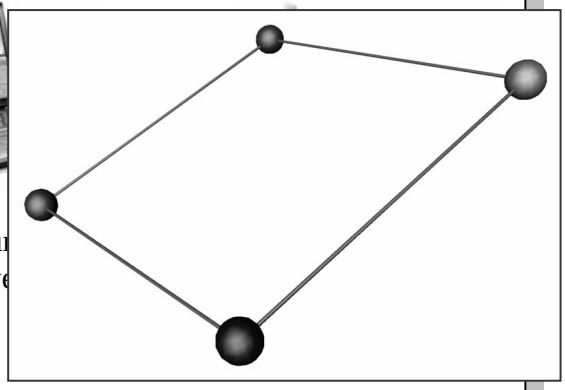
- Synchronisation entre les images virtuelles et le monde réel suffisamment fiable,
- Faible latence.

Au final, nous avons retenu la solution suivante pour un coût approximatif de 2200 Euros (ordinateur personnel non compris) :

Acquisition des données ➔ Traitement des données ➔ Affichage

Caméra iBot FireWire

Ordinateur
Apple Powe



Capteur InterTrax 2

Voici une description succincte du fonctionnement de la solution (la description détaillée du système et des composants sera donnée dans les chapitres suivants) :

L'utilisateur porte les lunettes Cy-Visor sur lesquelles on monte la caméra iBot FireWire et le capteur InterTrax 2. La transmission entre tous les composants se faisant par fils, l'ordinateur portable doit rester à portée de l'utilisateur, mais il faut noter que le système est complètement autonome :

- la caméra est le capteur sont alimentés depuis la batterie de l'ordinateur et les lunettes ont leur propre batterie,
- l'ensemble est peu encombrant et d'un poids acceptable (environ 4 Kg).

La caméra capture donc ce que verrait normalement l'utilisateur et envoie les images à l'ordinateur, lequel récupère également les informations du capteur. Un logiciel développé pour l'occasion traite ces données, calcule les images virtuelles et les superpose aux images réelles. Le résultat est ensuite affiché dans les lunettes.

Pour synchroniser les objets virtuels sur le monde réel, il est nécessaire d'établir un lien entre le monde

réel et le monde virtuel. Pour cela, on utilise un cadre spécial, avec des boules de couleurs ("marqueurs") dans ses coins (figure ci-contre) que l'on détecte avec la caméra. Ce cadre permet de définir un référentiel dans l'espace réel contre lequel est calibrée la synchronisation.
Une fois la calibration effectuée, on utilise les informations du capteur pour déterminer le mouvement de l'utilisateur et conserver la synchronisation. Lorsque cela est possible, on vérifie que la synchronisation est toujours correcte, en contrôlant par rapport au cadre.
Capteur et marqueurs sont donc complémentaires : les marqueurs permettent d'initialiser le capteur, tandis que le capteur permet de continuer à connaître l'orientation de la tête de l'utilisateur lorsque les marqueurs ne sont plus visibles.

Afin d'alléger le projet, nous avons dû effectuer certaines simplifications :

- Le capteur choisi ne fournit que 3 DDL (les 3 angles d'Euler). En effet, traquer l'utilisateur en 6 DDL impliquerait l'utilisation de bien plus de matériels et compliquerait nettement le système, d'autant plus que ce n'est pas nécessaire pour les exemples d'applications du prototype. Concrètement, cela signifie que **le système se désynchronisera si l'utilisateur se déplace et que les marqueurs sortent du champ de vision**. Lors de la mise en route du système, il est également indispensable que le cadre soit complètement dans le champ de vision de la caméra.
- Les lunettes 3D choisies sont monoscopiques (la même image est projetée sur les yeux gauche et droit). Il est donc impossible de donner à l'utilisateur une véritable impression de profondeur, mais comme il n'est pas prévu qu'il interagisse avec les objets distants réels ou virtuels, cette limite est acceptable. Il faut noter par ailleurs que les lunettes 3D stéréo avec une bonne résolution d'image sont extrêmement coûteuses.

PARTIE MATERIELLE DU PROJET

Nous allons maintenant étudier en profondeur la réalisation du projet. Dans cette première partie, nous présenterons en détail les composants matériels utilisés (caractéristiques, fonctionnement, problèmes lors de utilisation...) et justifierons leur choix.

Capteur InterSense InterTrax 2

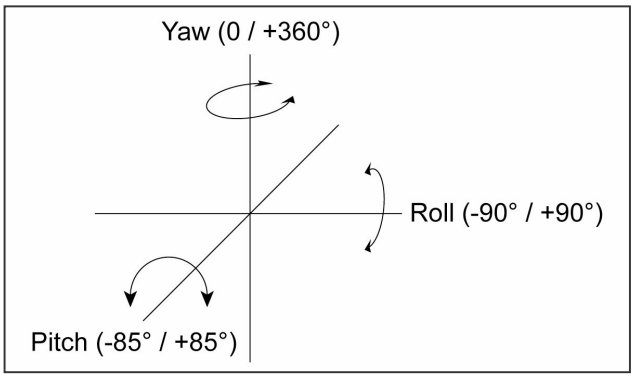
Présentation

L'InterTrax 2 est un capteur d'orientation en 3 DDL (angles d'Euler) conçu par la société américaine Intersense[1], spécialisée dans les capteurs pour la réalité virtuelle. Cette société a été crée en coopération avec le M.I.T. par un doctorant qui avait travaillé sur les capteurs de mouvements. Au moins deux brevets ont été déposés concernant la technologie mise en œuvre dans l'InterTrax 2 [1].

Ce capteur est l'un des derniers produits d'InterSense, qui oriente maintenant une partie de sa gamme de produits vers une utilisation "grand public".

Caractéristiques

La plage de mesure angulaire du capteur est indiquée sur le schéma ci-contre.



La documentation indique que l'InterTrax 2 serait extrêmement stable (aucune vibration dans la mesure), insensible aux perturbations électromagnétiques, n'aurait pas de dérive et utiliserait des algorithmes de prédiction pour compenser automatiquement sa propre latence.

[1] http://www.isense.com

Voici les caractéristiques du capteur extraites de la documentation :

Maximum Angular Rate	± 720° azimuth elevation ± 360° roll
Minimum Angular Rate	3° per second
Internal Update Rate	256 Hz
Internal Latency	4 milliseconds
Angular Resolution	0.02° relative
Size	9.4 x 2.7 x 2.7 cm
Weight	39 grams

Fonctionnement

InterSense reste très discret sur le fonctionnement précis de ses capteurs, sans aucun doute pour protéger son savoir-faire. Toutefois, en recoupant les informations des maigres documentations techniques, du site Internet et des brevets, il a été possible de rassembler les renseignements suivants :
L'InterTrax 2 est constitué de 8 capteurs reliés à un microprocesseur, lequel utilise des algorithmes sophistiqués pour déterminer son orientation dans l'espace. Ces capteurs sont des piezo-électriques, qui mesurent la vitesse rotationnelle, et des détecteurs du champ gravitationnel ou magnétique. Ces seconds sont utilisés pour compenser la dérive lors de l'intégration des vitesses.
Le capteur se calibre une première fois lorsqu'il est mis sous tension puis à intervals réguliers.

L'InterTrax 2, qui est destiné à être connecté aux ordinateurs personnels, utilise donc une interface USB. Cette interface a un débit maximal de 1.5 Mo/s[1], ce qui est plus que suffisant pour transmettre les informations du capteur avec une latence minimale.
Le capteur respecte la norme Human Interface Devices (HID) [2] qui définit un protocole de communication universel pour les périphériques USB. Les angles sont donc codés sur des valeurs 16 bits (0 à 65535) ce qui garantie une précision théorique de 5 millièmes de degré environ alors que le capteur a une précision effective de 2 centièmes de degré.
L'InterTrax 2 s'alimente directement sur le bus USB (5V à 500mA disponible).

[1] Pour l'ensemble des périphériques connectés sur un port.

Conclusion

Nous avions choisi ce capteur à cause de sa taille, de sa connexion USB, et sur la base de ses caractéristiques flatteuses.
Son utilisation a montré qu'il tient ses promesses : il est extrêmement précis, a une latence très faible et ses mesures sont très stables. Nous n'avons trouvé qu'un seul point faible : si on l'oriente vers le zénith (Roll -> 90°), qu'on le dépasse, et qu'on revient vers la position initiale, il peut perdre ses références et retourner ensuite des angles erronés. Mais généralement, il finit par se recalibrer correctement.

ADDENDUM : *Lors de l'implémentation de la phase finale du projet, c'est-à-dire lors de la fusion entre les informations du capteur InterTrax 2 et celles des marqueurs, pour une raison inconnue, l'InterTrax 2 a commencé à présenter une très forte dérive / hystérésis : si on le tourne puis on le fait revenir à sa position initiale, il peut y avoir jusqu'à des dizaines de degrés d'offset. Dans le laps de temps restant, il n'a pas été possible d'en déterminer la cause ou d'y trouver un remède.*

Caméra FireWire iBot

Description

Nous avons choisi d'utiliser comme caméra une simple webcam pour ordinateurs personnels (l'iBot d'Orange Micro[1]), mais qui a la particularité avoir une connexion FireWire (50 Mo/s), alors que toutes les webcams actuelles se branchent sur le bus USB (1.5 Mo/s). Disposer d'une telle vitesse de transmission permet à la caméra d'utiliser un capteur CCD en 640x480 au lieu des 320x240 habituels sur de telles caméras, et de transmettre les images quasiment sans les compresser. Par ailleurs, la transmission des données numériques se fait sur des câbles blindés, en mode différentiel, pour assurer une insensibilité maximale au bruit.
En plus, l'iBot s'alimente directement sur le bus FireWire (12 V à 500 mA disponible).

Comparée à une caméra analogique qui serait connectée à une carte d'acquisition vidéo, cette solution entièrement numérique est bien plus simple à utiliser et garantit une qualité d'image idéale. Par ailleurs, l'iBot ne capture pas un flux vidéo interlacé, mais une série d'images fixes, ce qui garantit une qualité maximale pour chaque image.

Fonctionnement

L'iBot est construite principalement autour d'un capteur CCD 640x480 et d'un circuit Texas Instrument TSB15LV01 [3]. Ce circuit est conçu spécialement pour l'implémentation de caméras connectées au bus FireWire : il prend en charge la CCD, tous les réglages de façon automatique ou manuelle (luminosité, exposition, netteté, saturation, la balance des blancs, le gamma…), la gestion du protocole FireWire et la transmission des données, et peut même contrôler un moteur de zoom / mise au point (non présent dans l'iBot).

Une optique très simple avec une mise au point manuelle est montée devant le capteur CCD.

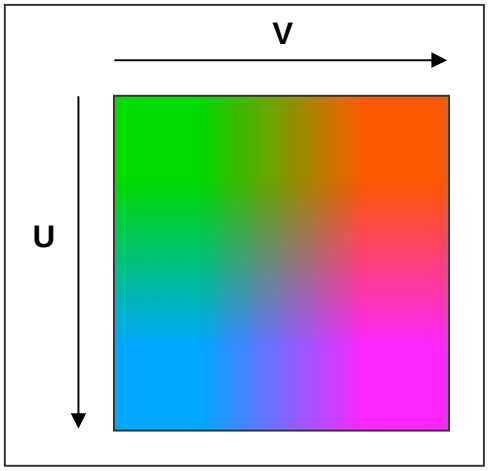
[1] http://www.orangemicro.com

Le contrôleur TI supporte les formats suivants :

MODE	VIDEO FORMAT	FRAME RATE (Frame per Second)			
		30	15	7.5	3.75
Mode_0	160 x 120 YUV(4:4:4) 24 bits/pixel	1/2 L 80 P 60 Q	1/4 L 40 P 30 Q	1/8 L 20 P 15 Q	
Mode_1	320 x 240 YUV(4:2:2) 16 bits/pixel	1 L 320 P 160 Q	1/2 L 160 P 80 Q	1/4 L 80 P 40 Q	1/8 L 40 P 20 Q
Mode_2	640 x 480 YUV(4:1:1) 12 bits/pixel	2 L† 1280 P 480 Q	1 L 640 P 240 Q	1/2 L 320 P 120 Q	1/4 L 160 P 60 Q
Mode_3	640 x 480 YUV(4:2:2) 16 bits/pixel		1 L† 640 P 320 Q	1/2 L 320 P 160 Q	1/4 L 160 P 80 Q
Mode_4	640 x 480 RGB 24 bits/pixel		1 L† 640 P 480 Q	1/2 L 320P 240Q	1/4 L 160P 120Q
Mode_5	640 x 480 Y (Mono) 8 bits/pixel	2 L† 1280P 320 Q	1 L 640 P 160 Q	1/2 L 320 P 80 Q	1/4 L 160 P 40 Q

Le format RGB est totalement non-compressé : chaque pixel est transmis sur 24 bits avec 8 bits pour chaque composante de couleur Rouge / Vert / Bleu. Dans un tel cas, en 640x480, il faut transmettre 900 Kb / image et le contrôleur TI limite en conséquence le nombre d'images par seconde à 15 soit environ 13.5 Mo/s. Il est vrai que le bus FireWire supporte 50 Mo/s, mais l'ordinateur doit aussi pouvoir traiter toutes les données en temps réel.

Le format YUV, très utilisé en vidéo, transmet les images dans un autre espace de couleur que le RGB. Dans cet espace, l'information de couleur (UV) est séparée de l'information de luminosité (Y). L'image ci-contre présente l'aspect du spectre UV pour une valeur de luminosité moyenne.



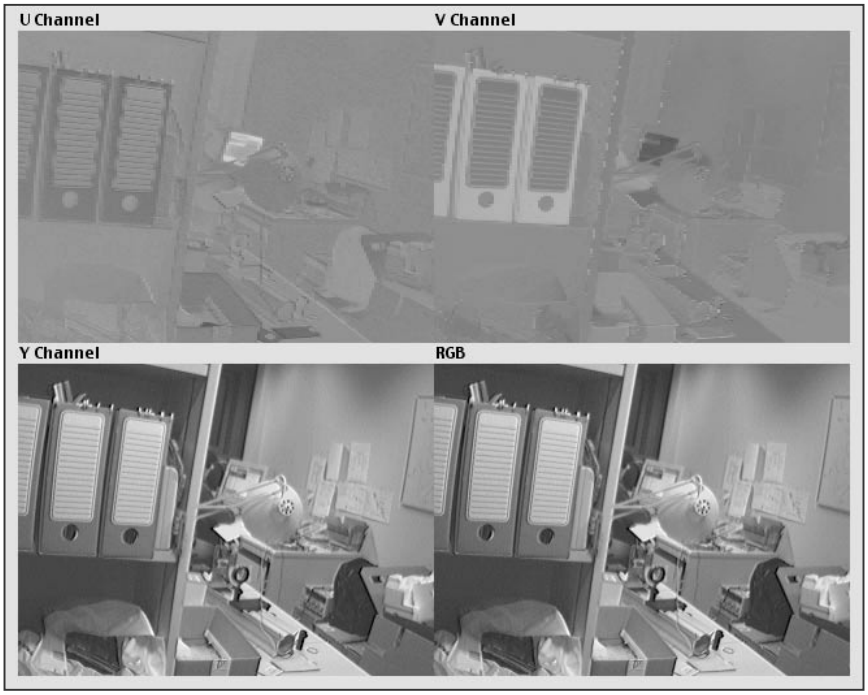
Dans les tailles d'image qui nous intéressent, le contrôleur TI transmet le format YUV en compressé 4:2:2 ou 4:1:1. Cela signifie qu'il y a 2 fois moins ou 4 fois moins d'information de couleur que d'information de luminosité, la luminosité n'étant pas du tout compressée. Par exemple, pour image en 320x240 YUV(4:2:2), les pixels adjacents sont transmis 2 à 2 par paquets de 32 bits décomposés en 4 blocs de 8 bits (U Y_1 V Y_2). L'image est en fait transmise sous la forme d'une image 320x240 en Y et 160x240 en UV. Cela équivaut à une compression spatiale : 1 pixel de couleur pour 2 pixels de luminosité.

Le mode YUV compressé joue sur le fait que l'œil est plus sensible à la luminosité qu'à la couleur, et permet donc une compression simple et efficace des images.

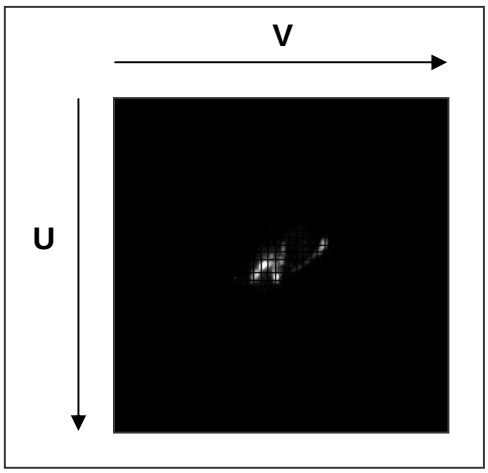
Le format Y est totalement non-compressé : chaque pixel est transmis sur 8 bits, mais ne contient que les informations de luminance (image en niveaux de gris).

Caractéristiques du signal vidéo

L'image ci-dessous, tirée de la caméra, est affichée sous forme de ses composantes U, V et Y, puis reconstituée dans l'espace RGB :
L'étude des images fournies par la caméra, comme on le voit dans l'exemple ci-dessus, révèle que le canal Y utilise bien toute la dynamique possible (plage de valeurs de 0 à 255), mais que les canaux U et V apparaissent ternes et peu contrastés.



Si l'on examine le spectre UV de cette même image (figure ci-contre), on remarque les points suivants :



- Seule une très faible surface du spectre est utilisée.
- Même au sein de la zone utilisée dans le spectre, il existe des "vides" : il existe des lignes horizontales et verticales dans l'espace UV où aucun pixel ne se trouve.
- Modifier le réglage de saturation sur la caméra ne fait qu'agrandir artificiellement le spectre, sans augmenter sa résolution (les "vides" se font de plus en plus importants).

On peut avancer comme explication que l'échantillonnage de la chrominance se ferait sur moins de 8 bits, et que le réglage de la saturation aurait lieu en aval de l'échantillonnage et donc sur des valeurs discrètes. Modifier la saturation ne permet donc pas d'influencer la résolution ou la plage de la chrominance.

Caractéristiques optiques

Afin de pouvoir superposer correctement les images virtuelles sur les réelles, il est nécessaire que la caméra virtuelle (utilisée pour calculer les images 3D) ait les mêmes paramètres que la caméra physique (notamment l'angle d'ouverture).
Par ailleurs, une webcam, même haut de gamme, est loin de ressembler au modèle théorique d'une caméra pinhole (projection à travers un diaphragme infiniment petit) car son objectif presque grand angle déforme sérieusement l'image. Il est nécessaire de caractériser cette distorsion avec exactitude.

Pour déterminer les paramètres de la caméra, on utilise toolkit de calibration MathLab [4] auquel on fournit une série d'images d'une plaque de calibration connue. Ces images sont obtenues par la caméra selon un maximum d'angles de vue (voir image ci-contre). Dans notre cas, la plaque est une grille de 10x10 carrés blancs de 1 cm de côté, répartis uniformément sur un fond noir.

Nous obtenons les résultats suivants (valeurs en pixels selon les axes X et Y):

- Focale : F_x=731.413±15.39 et F_y=725.256±11.66
- Centre optique : C_x=365.906±5.201 et C_y=191.504±16.45
- Coefficient de distorsion radial : R_x=-0.3847±0.02482 et R_y=0.159±0.09803
- Coefficient de distorsion tangentiel : T_x=0.009642±0.008039 et T_y=0.003334±0.001131

La *focale* représente la distance entre le point focal et le plan de projection. Le *centre optique* correspond au centre de projection de l'image, qui n'est pas le milieu géométrique des images capturées (en [320,240] pour des images en 640x480). Les *coefficients de distorsion radial et tangentiel* permettent de définir un champ vectoriel de la distorsion dans l'image.

Connaissant la focale et les dimensions du plan image, nous pouvons calculer l'angle d'ouverture théorique de la caméra iBot :

$$\alpha_x = 2 \times \tan^{-1}\left(\frac{640}{F_x}\right) = 82.37°$$

Nous sommes également en mesure de calculer la position avec distorsion (X,Y) d'un point réel (x,y) dans le plan image :

*Eq.*1
$$\begin{pmatrix} x_h \\ y_h \\ 1 \end{pmatrix} = \begin{bmatrix} F_x & 0 & C_x \\ 0 & F_y & C_y \\ 0 & 0 & 1 \end{bmatrix}^{-1} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{x - C_x}{F_x} \\ \frac{y - C_y}{F_y} \\ 1 \end{pmatrix}$$

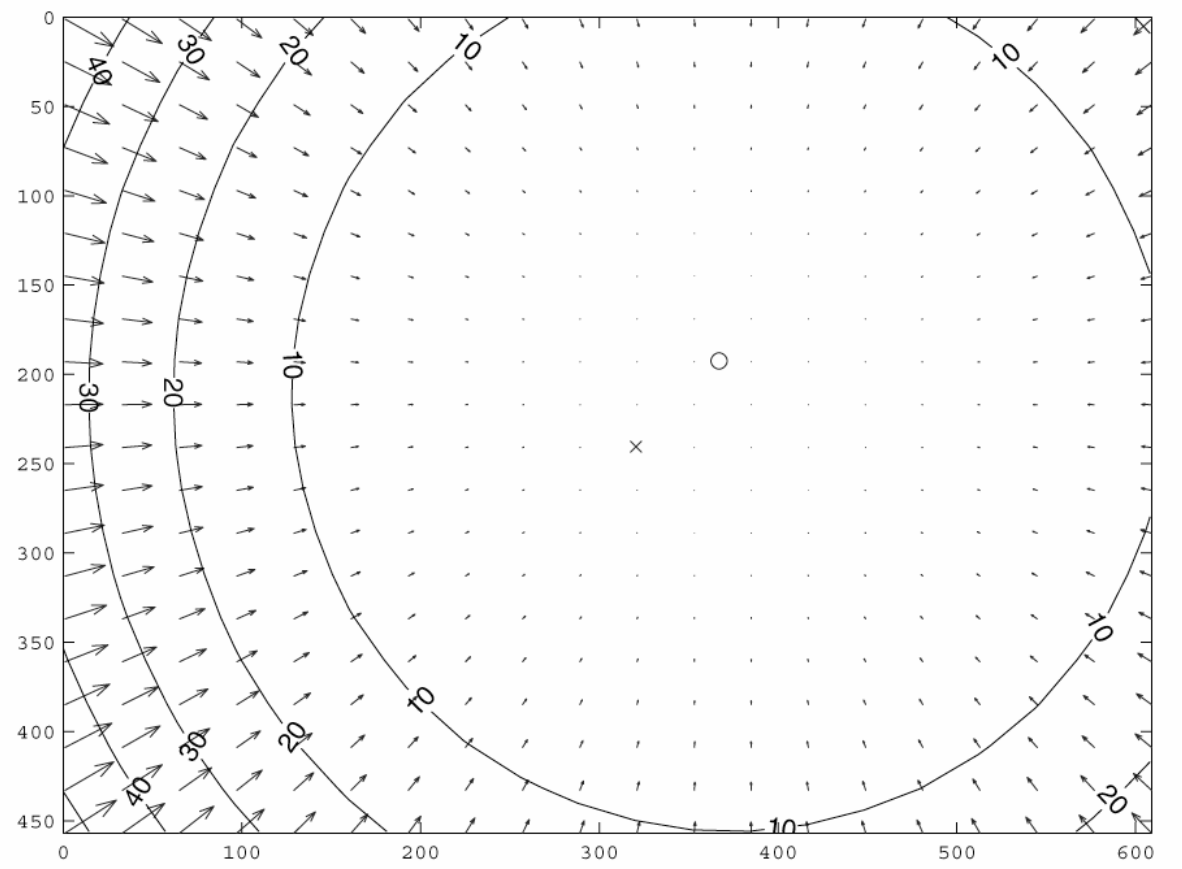
*Eq.*2
$$\begin{cases} X_h = x_h + x_h \cdot \left(R_x \cdot r^2 + R_y \cdot r^4\right) + \left[2 \cdot T_x \cdot x_h \cdot y_h + T_y \cdot \left(r^2 + 2 \cdot {x_h}^2\right)\right] \\ Y_h = y_h + y_h \cdot \left(R_x \cdot r^2 + R_y \cdot r^4\right) + \left[2 \cdot T_y \cdot x_h \cdot y_h + T_x \cdot \left(r^2 + 2 \cdot {y_h}^2\right)\right] \end{cases} \quad avec \quad r^2 = {x_h}^2 + {y_h}^2$$

*Eq.*3
$$\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = \begin{bmatrix} F_x & 0 & C_x \\ 0 & F_y & C_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} X_h \\ Y_h \\ 1 \end{pmatrix} = \begin{pmatrix} X_h \cdot F_x + C_x \\ Y_h \cdot F_y + C_y \\ 1 \end{pmatrix}$$

(x,y) et (X,Y) sont exprimées en pixels par rapport au coin supérieur gauche de l'image.
(x_h,y_h) et (X_h,Y_h) sont des coordonnées "pseudo-homogènes" sans unités exprimées par rapport au centre de l'image et indépendantes des différences de focale sur les axes X et Y.

Pour corriger la distorsion, il faut donc inverser l'équation 2 une fois les paramètres déterminés. Comme il n'existe pas de solution analytique, on doit faire des résolutions numériques de ce système.

Les équations de la page précédente nous permettent de représenter la distorsion dans tout le plan image (en 640x480):
On remarque que la distorsion est très importante dans les coins gauches de l'image. Notamment,



les pixels dans le coin supérieur gauche se trouvent 40 pixels plus vers l'intérieur de l'image que si la caméra n'avait pas eu de distorsion.

Conclusion

La caméra iBot se présente comme un très bon compromis qualité / prix eu égard sa taille : les images sont d'une très bonne qualité et très fluides. Ses points faibles sont sa faible dynamique des couleurs en mode YUV et surtout sa distorsion optique ; deux inconvénients qui auront des conséquences sur la calibration du système de réalité augmentée. Par contre la qualité des images et leur fluidité (on obtient sans problème 30 images / seconde en 320x240 YUV(4:2:2)) ne fait aucun doute.

Lunettes 3D Cy-Visor 4400VP

Présentation

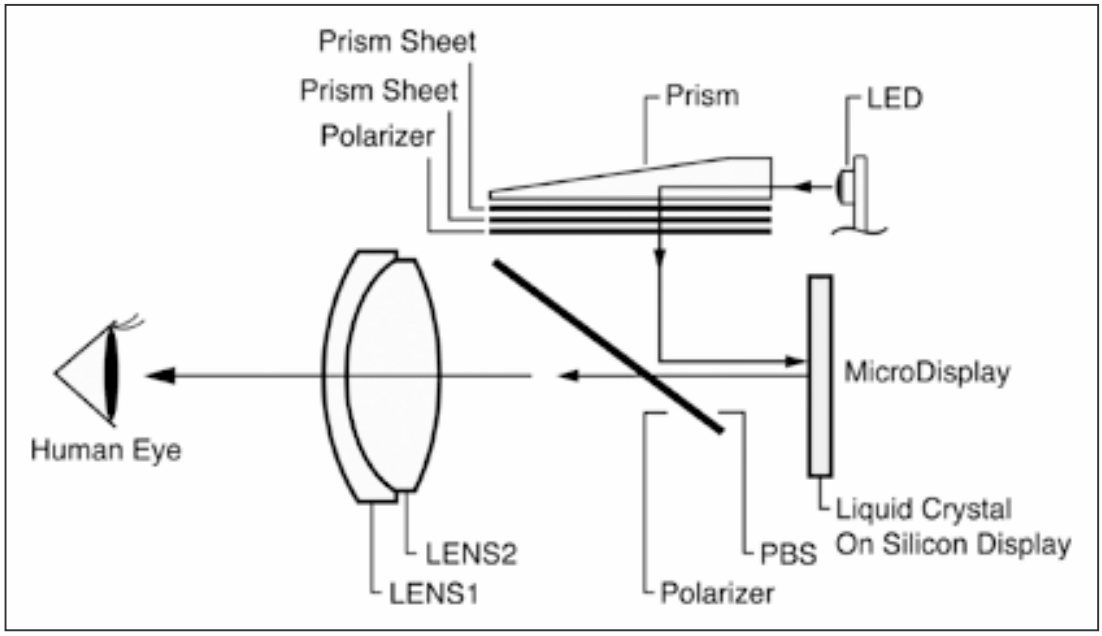
Nous avons choisi d'utiliser les lunettes Cy-Visor 4400VP car elles ont la plus grande résolution que nous avons pu trouver et pour un prix abordable. Leurs caractéristiques sont les suivantes :

- Affichage : 2 LCD de 125 mm de diagonale
- Résolution : 800x600 en millions de couleurs
- Image virtuelle : 112 cm à 2 m
- Angle de vue : 31°
- Connectique variée : SVHS, D-sub 15pin, RCA vidéo
- Formats vidéos : Pal, Secam, NTSC, VGA, SVGA…
- Alimentation : sur secteur ou batterie

Fonctionnement

La seule information disponible concernant le fonctionnement du Cy-Visor est un schéma extrait de la documentation :

Utilisation



Même si les lunettes 3D ont fait de gros progrès ces dernières années, le système est loin d'être parfait. L'image du Cy-Visor est effectivement très bonne et les couleurs acceptables, mais il reste des défauts conséquents :

- L'ergonomie générale est très moyenne : le casque prend appuie sur le front et est maintenu par un "bandeau" que l'on serre autour de la tête. Le système étant en porte-à-faux, on doit le serrer très fort et l'ensemble devient rapidement inconfortable. En fait il est plus simple et plus agréable de le tenir à la main devant les yeux.
- Les câbles entravent les mouvements et pèsent sur le casque. Le câble du Cy-Visor seul est certes très mince, mais il faut lui adjoindre les câbles du capteur USB et de la caméra FireWire (épais et rigide à cause du blindage).
- Du fait de l'angle de vue restreint (31°), on voit un cadre noir important autour de l'image : on a plus l'impression de se trouver dans une salle de cinéma qu'immersé dans un monde de réalité augmenté.
- L'immersion est également pénalisée par la difficulté de s'isoler du monde extérieur : les caches noirs fournis avec les lunettes et qui se placent entre les écrans et les yeux s'adaptent mal aux différentes morphologies des utilisateurs. Pour qu'ils soient réellement efficaces, il faut presser les lunettes contre les yeux, ce qui n'est pas confortable car les caches sont assez durs.
- Fréquence de rafraîchissement des écrans LCD insuffisante (autour de 50 Hz): on observe un scintillement des couleurs dans l'image lorsque l'on tourne la tête un peu rapidement[1].

Conclusion

Les Cy-Visor ont une allure futuriste, affichent une bonne image, mais il n'est malheureusement pas réaliste d'envisager de porter un tel système pour des périodes supérieures à quelques dizaines de minutes.

[1] Un scintillement peut apparaître même sans tourner la tête, mais il s'agit alors d'un problème de synchronisation du rendu 3D avec le rafraîchissement du signal vidéo qui sort de l'ordinateur. Ce problème est en fait un bug dans les drivers 3D de la carte vidéo.

Ordinateur Personnel Apple PowerBook G3 "Pismo"

Présentation

Disposant déjà à titre personnel d'un PowerBook G3 qui dispose de toutes les caractéristiques requises pour le projet, nous avons décidé de l'utiliser pour le système de réalité augmentée.

Caractéristiques principales du PowerBook :

- Processeur PowerPC G3 400 MHz
- bus 100 MHz
- 1Mb de cache L2, 384 Mo de RAM
- Disque dur 20Gb, lecteur de DVD
- Écran LCD TFT 14" 1024x768
- Carte vidéo ATI Rage 128 AGP 8 Mo
- 2 ports FireWire alimentés
- 2 ports USB alimentés
- Sortie vidéo en D-sub 15pin ou SVHS pour connecter un écran supplémentaire qui affiche une image distincte de celle de l'écran LCD
- Système d'exploitation : MacOS 9.1 US

Fonctionnement

L'ordinateur constitue le cœur du système : la caméra iBot est connectée sur l'un des ports FireWire tandis que le capteur InterTrax 2 est connecté à l'un des ports USB. L'ordinateur héberge le programme de réalité augmentée et affiche le résultat sur le Cy-Visor, lequel est connecté sur la sortie vidéo SVHS[1] réglée en 640x480 Pal 50 Hz milliers de couleurs (16 bits).

Conclusion

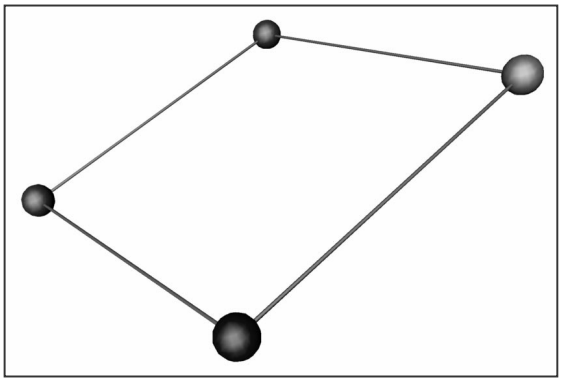
Comme son utilisation l'a montrée, la puissance de calcul de ce portable ainsi que les performances de sa carte vidéo sont amplement suffisantes pour le prototype de réalité augmentée. Comme nous le verrons dans le chapitre suivant, les limites de performances viendront de facteurs logiciels, car il a fallu donner la priorité à l'universalité du code et à sa facilité de réutilisation, plutôt qu'aux performances.

Par contre, nous n'avons pas une qualité d'image optimale à cause des différentes résolutions vidéos utilisées par chacun des périphériques : la caméra capture en 320x240 ou 640x480, la sortie écran est en 640x480, mais est convertie en un signal Pal dont le standard est environ 720x576, avant d'être enfin affiché sur le Cy-Visor en 800x600. Par contre, l'ensemble de ces conversions lisse l'image finale (interpolation bilinéaire) et lui évite d'apparaître pixelisée.

[1] À cause de la façon dont le Cy-Visor gère ses entrées vidéo, cette sortie est plus simple à utiliser que la sortie moniteur DB-15, même si en théorie la qualité d'image est moins bonne.

Le cadre de calibration

Introduction



Pour synchroniser les images virtuelles sur les réelles, il est nécessaire de disposer d'un objet "point de repère" dans le monde réel à partir duquel on puisse définir un référentiel, soit 3 axes orthonormés.
Il existe de nombreuses façons de construire un tel objet : on peut utiliser une pyramide, un cube, 3 tiges perpendiculaires deux à deux…

Nous avons choisi d'utiliser 4 boules disposées sur un cadre carré (ou rectangulaire) car :

- la structure est simple à construire et légère,
- cette structure est très bien adaptée aux exemples d'applications du prototype de réalité augmentée,
- comme nous le verrons plus loin, il existe une méthode très simple pour déterminer la position d'un cadre dans l'espace suivant son image sur la caméra, alors que les méthodes génériques pour un ensemble quelconque de points sont très complexes[1],
- les boules ont le même aspect quel que soit le point de vue, ce qui facilite leur détection visuelle.

Construction

Nous avons réalisé le cadre à partir des éléments suivants:

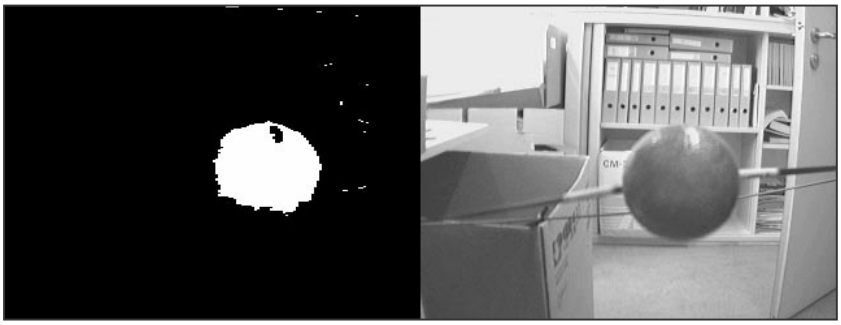
- 4 tubes en carbone de 6 mm de diamètre externe et de 50 cm de long,
- 4 boules de sagex de 5 cm de diamètre,
- 4 coudes en laiton - chassés entre deux des tubes pour former un coin.

Les couleurs des boules doivent être choisies pour être facilement détectable dans le spectre UV, tout en étant peu communes dans l'environnement pour éviter que des objets du décor ne soient confondus avec les boules. Par ailleurs, avoir une couleur par boule est l'idéal car il n'y a aucune singularité possible pour déterminer l'orientation exacte du cadre dans l'espace.
Nous avons choisi comme compromis d'utiliser 3 couleurs se situant aux extrémités du spectre UV: rouge, vert et bleu. Cela signifie qu'il existe une singularité au niveau des deux boules qui sont de couleur identique, mais elle est facile à résoudre.

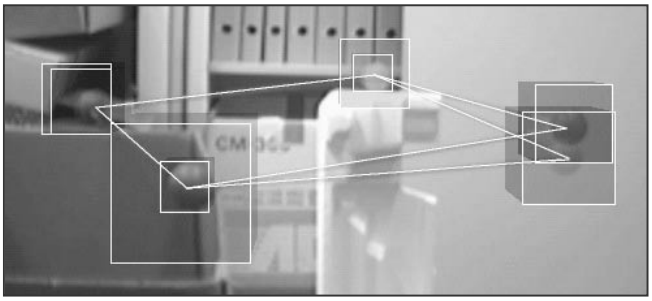
[1] Dans un tel cas, on utilise typiquement une méthode de "best fit" qui consiste à trouver la meilleure position dans l'espace de l'ensemble de points, telle que sa projection se rapproche le plus possible de la projection observée.

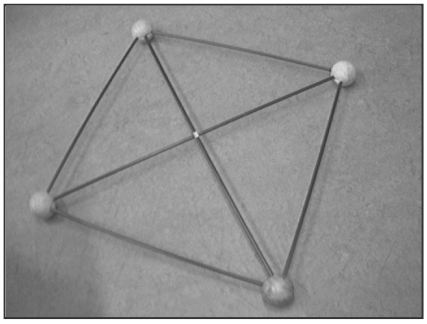
Pour faciliter la détection d'une boule dans l'environnement, il faut que sa couleur soit la plus unie possible. Or deux éléments viennent la perturber : les ombres, et les réflexions spéculaires de la surface. Comme on le voit sur l'image ci-dessous, les algorithmes de détection sont alors leurrés, et l'objet détecté ne ressemble plus vraiment à une boule (voir image ci-dessous).
On ne peut pas faire grand chose pour éliminer les ombres sur la surface à moins d'avoir un éclairage parfaitement omnidirectionnel.
Pour diminuer les reflets sur les boules, on pense d'abord à utiliser une surface rugueuse (ce qui est déjà le cas avec du sagex) et la plus mate possible. En fait, on ne peut jamais supprimer totalement la spécularité et au final, on ne fait qu'agrandir la taille du reflet même si son intensité diminue. Il est plus efficace d'effectuer l'opération inverse et d'augmenter la spécularité pour

avoir le reflet le plus concentré possible, même s'il est très intense. Nous avons donc verni les boules : l'image suivante montre bien que s'il reste toujours le problème de l'ombre, les réflexions spéculaires sont réduites au simple reflet des néons et la forme détectée est plus proche d'une boule.



Par ailleurs, le cadre réalisé nous a posé problème au bout de plusieurs semaines car les boules n'étaient plus coplanaires (d'environ 1 à 2 cm) et cela influence considérablement les algorithmes de calibration, comme on le voit sur l'image ci-dessous, où, dans le cas de non-coplanarité, les tailles des cubes virtuels superposés sur les boules réelles deviennent aberrantes :
Il a donc été nécessaire de reconstruire un cadre nettement plus résistant. Cette fois-ci, nous





avons utilisé des tubes de 8 mm (6 mm de diamètre intérieur), connectés à des équerres[1] par des goupilles, le tout chassé ou collé. Pour résister à la torsion, nous avons placé d'autres tubes de même diamètre en diagonal. Enfin, nous avons utilisé des boules de sagex de 6 cm de diamètre.

Conclusion

Utiliser un cadre comme référentiel de calibration est la meilleure solution pour notre prototype car c'est facile à construire et l'on peut en fabriquer de toutes les dimensions.
Pour certaines applications de réalité augmentée (si l'on veut afficher des objets virtuels sur une surface plane par exemple), placer 4 marques de couleurs en carré ou rectangle est une solution très simple qui demande très peu de modification de l'environnement et est rapide à mettre en place.
Par contre, pour d'autres applications (par exemple dans le domaine médical), il est évident que cette méthode n'est pas envisageable : on ne peut demander au chirurgien de fixer un cadre de 30x30 cm solidement sur le patient.

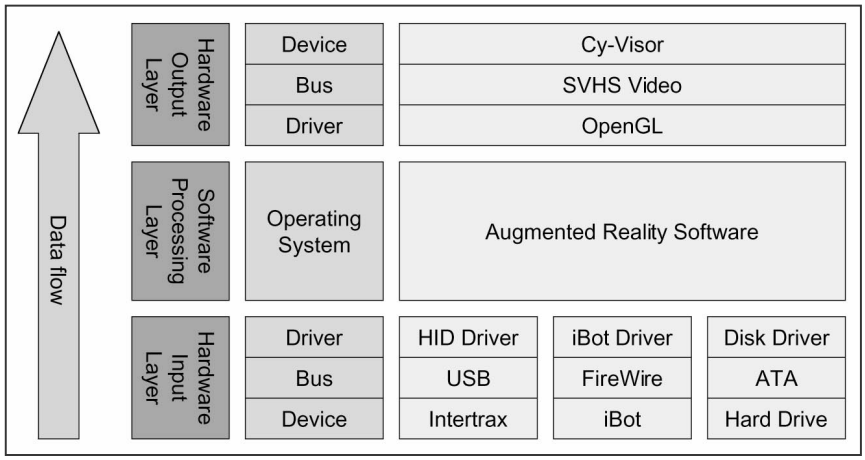
[1] En aluminium, usinés pour l'occasion.

PARTIE LOGICIELLE DU PROJET

Nous allons maintenant étudier la réalisation du logiciel de réalité augmentée, qui représente le cœur du système.

Schéma d'ensemble

Le schéma ci-dessous représente la partie informatique du projet c'est-à-dire la conception d'un logiciel de réalité augmentée :
Le schéma permet de voir l'intégration du logiciel développé pour le projet dans le système informatique et le chemin parcouru par les données (du bas vers le haut). Ce logiciel est une boîte

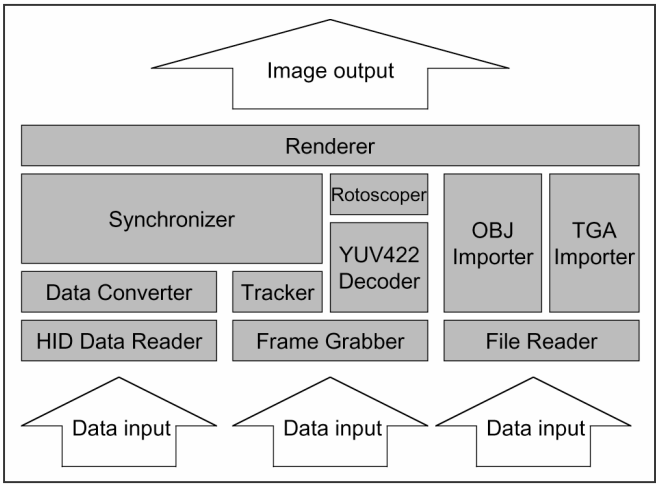


noire dont les entrées sont connectées aux drivers des interfaces de l'ordinateur (USB, FireWire, disque), et la sortie à l'affichage OpenGL.
Les données d'orientation, de vision et des objets 3D sont récupérées sur chacune des interfaces et interprétées par le logiciel. Il calcule ensuite l'image finale qui est dessinée par OpenGL sur la sortie vidéo connectée aux lunettes 3D.

Les deux contraintes essentielles lors du développement d'un logiciel informatique sont la portabilité de son code, et sa facilité de réutilisation dans d'autres projets. La solution adoptée pour le logiciel de réalité augmentée est donc une approche modulaire sous forme de "composants logiciels", indépendants les uns des autres, comme autant de boîtes noires qui ne communiqueraient que par leurs entrées / sorties.
Une telle solution est très efficace car, d'une part le portage du code sur un autre système d'exploitation est facilité (le code dépendant du système d'exploitation ne se trouve que dans certains composants spécifiques), et d'autre part, il est facile d'utiliser le code d'un composant "universel" (qui ne dépend d'aucune technologie propriétaire) dans un autre projet. La lecture et la compréhension du code source en sont aussi grandement facilitées.

Le schéma ci-dessous montre en détail la construction du logiciel de réalité augmentée sous forme de "couches" de composants :
Voici un descriptif succinct de chaque composant logiciel :

- Composants spécifiques à la plateforme (ils doivent êtres réécrits pour que le programme fonctionne sur un autre système d'exploitation que MacOS):
  - **HID Data Reader**: récupère les données non décodées du capteur InterTrax 2 sur le bus USB.



  - **Frame Grabber**: récupère les images non-décompressées de l'iBot sur le bus FireWire.
  - **File Reader**: charge en mémoire le contenu d'un fichier sans l'interpréter.
- Composants universels (ils n'utilisent aucune technologie propre à une plateforme donnée et leur code est donc en théorie compilable sur n'importe quel système d'exploitation):
  - **Data Converter**: convertit les données "brutes" de l'InterTrax 2 dans une unité standard (degrés).
  - **Tracker**: cherche des marqueurs de couleurs répondant à certaines caractéristiques dans une image encodée en YUV(4:2:2).
  - **YUV422 Decoder**: décompresse une image YUV(4:2:2) en une image RGB.
  - **OBJ Importer**: charge un objet en 3D depuis un fichier au format Alias WaveFront OBJ.
  - **TGA Importer**: charge une image depuis un fichier au format Targa TGA.
  - **Synchronizer**: effectue la synchronisation entre les images virtuelles et réelles à partir des informations du capteur et l'image des boules du cadre.
  - **Rotoscoper**: affiche l'image vidéo du monde réel en fond du rendu 3D.
  - **Renderer**: calcule et superpose le rendu 3D des objets virtuels sur l'image vidéo réelle.

Format du code source

Toute la partie logicielle a été développée sous CodeWarrior 7.0 de MetroWerks[1] en C (afin de faciliter la compréhension et le portage), mais avec des compilateurs C++ car ils sont plus rigoureux sur la syntaxe tout en offrant plus de possibilités.

[1] http://www.metrowerks.com

Le code de chaque composant se trouve dans un fichier source indépendant, accompagné de son fichier header.

La description et l'explication du code source est incluse dans le code même.

Implémentation des composants spécifiques

HID Data Reader

Ce composant a deux fonctions :

- trouver l'InterTrax 2 sur le bus USB et établir une connexion,
- recevoir les données de l'InterTrax 2 et en extraire les informations requises, en l'occurrence les 3 angles d'Euler en valeurs brutes.

InterSense ne fournit pas de driver MacOS pour l'InterTrax 2, mais comme ce dernier respecte la norme HID, c'est le driver générique HID de MacOS qui va le prendre en charge. Le composant va donc interroger ce driver pour savoir s'il gère un périphérique USB dont les codes produit (0x0001) et vendeur (0x0A94) correspondent à ceux de l'InterTrax 2. Si oui, il établit une connexion permanente avec le driver afin de recevoir les rapports envoyés régulièrement par le capteur.

Ces rapports sont des blocs de données de quelques centaines d'octets, dont le format répond au normes HID, et qui sont émis tous les 10 ms. Voici le "HID Report Descriptor" du capteur, décodé depuis l'hexadécimal :

```
05 01                Usage Page (Generic Desktop)
09 04                Usage (Joystick)
A1 01                Collection (Application)
17 00 00 00 00            Logical Minimum (0)
27 FF FF 00 00            Logical Maximum (65535)
75 10                     Report Size (16)
95 03                     Report Count (3)
09 33                     Usage (Rx)
09 34                     Usage (Ry)
09 35                     Usage (Rz)
81 02                     Input (...)
05 09                     Usage Page (Button)
15 00                     Logical Minimum (0)
01                        ?
35 00                     Physical Minimum (0)
45 01                     Physical Maximum (1)
75 01                     Report Size (1)
95 06                     Report Count (6)
19 01                     Usage Minimum (0)
29 06                     Usage Maximum (6)
81 02                     Input (...)
95 01                     Report Count (1)
75 02                     Report Size (2)
81 01                     Input (...)
C0                   End Collection
```

Les 3 angles d'Euler sont donc les valeurs des champs R_x, R_y et R_z du rapport qui sont codés sur 16 bits en non-signé, soit de 0 à 65'535. Dans le cas de l'InterTrax 2, R_x correspond au Yaw, R_y au Roll et R_z au Pitch.

Le composant utilise des routines de l'API USB de MacOS pour extraire ces valeurs du rapport et les stocke dans des variables globales. Ces variables sont donc mises à jour en permanence et automatiquement tous les 10 ms.

Frame Grabber

Parmi tous les modes de capture de l'iBot (cf. section consacrée à la caméra FireWire iBot dans le chapitre Partie matérielle), nous n'en avons retenu que deux:

- 320x240 à 30 images / seconde en YUV(4:2:2)
- 640x480 à 15 images / seconde en YUV(4:2:2)

En effet, le format YUV(4:2:2) est le seul disponible en 320x240, et en 640x480 le format RGB nécessite de gérer beaucoup plus de données (24 bits / pixel contre 16 en YUV) pour une qualité d'image quasi-similaire pour l'œil. Par ailleurs, le format YUV possède un avantage important pour la détection des marqueurs de couleurs, car les informations de couleur et de luminosité sont séparées : on peut donc directement chercher les marqueurs dans l'espace UV de l'image.

Nous avons utilisé deux méthodes pour récupérer les images compressées depuis l'iBot :

- La première (plutôt de bas niveau) a été d'établir une connexion directe avec le driver de l'iBot pour avoir des performances maximales. Dans ce cas, le composant doit demander au driver de capturer une image dès qu'une nouvelle est requise et attendre en asynchrone que les données arrivent (avec bien entendu la limite de 30 images par seconde). Cette approche s'est avérée instable dans certains cas (plantages) et n'a pas été approfondie.
- La deuxième (de haut niveau) est d'utiliser QuickTime, le système multimédia de MacOS, pour effectuer la capture. L'avantage, c'est que le composant peut alors utiliser n'importe quelle caméra compatible QuickTime, et peut afficher une interface utilisateur qui permet facilement de la configurer. Avec cette méthode, QuickTime capture automatiquement un maximum d'images par seconde, et appelle une routine "callback" installée par le composant dès qu'une image est capturée, et lui passe les données.

Les données récupérées (environ 300 Ko par image) sont directement au format YUV(4:2:2) et sont copiées dans une variable globale pour être facilement accessibles par les autres composants.

Comme la détection des marqueurs est liée à la couleur, il est indispensable que les couleurs de la caméra soient stables. Il faut donc désactiver tous les systèmes automatiques : gain, temps d'exposition, balance des blancs, luminosité, saturation… et leur donner des valeurs constantes. L'inconvénient est que ces valeurs doivent être déterminées suivant l'environnement (intérieur ou extérieur, jour ou nuit).
Le système n'est donc pas adaptatif : il doit fonctionner dans des conditions connues à l'avance.

File Reader

Ce composant extrêmement simple se contente d'utiliser les API de MacOS pour charger le contenu d'un fichier sur le disque dans un bloc en mémoire.

Implémentation des composants universels

Data Converter

Ce composant convertit simplement les valeurs angulaires brutes retournées par l'InterTrax 2 (entre 0 et 65535) en degrés. Les formules de conversions sont donc de simples règles de trois :

$$R_x^{\deg} = R_x^{abs} \times 360 \div 65535$$
$$R_y^{\deg} = R_y^{abs} \times 180 \div 32678$$
$$R_z^{\deg} = R_z^{abs} \times 180 \div 32678$$

Pour que les conversions de R_y et R_z fonctionnent correctement, il faut "typecaster" avec le compilateur les valeurs brutes en 16 bits signées, alors qu'elles sont à l'origine des valeurs 16 bits non signées.

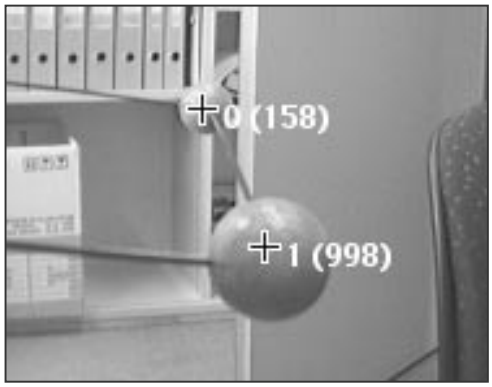
Les valeurs retournées par le composant sont donc bien dans les plages requises : 0° / 360° ou –180° / +180° comme l'indique la documentation du capteur (cf. la section sur l'InterTrax 2 au chapitre Partie matérielle).

Tracker

Le rôle du composant "Tracker" est de repérer et suivre des marqueurs (ou plus exactement leurs centres) sous forme de boules de couleurs dans des images vidéos. Nous avons choisi d'utiliser directement les images dans leur format natif YUV car, d'une part, cela évite une conversion en RGB, et d'autre part, la détection de couleur est facilitée par le fait que luminosité et chrominance sont séparées.

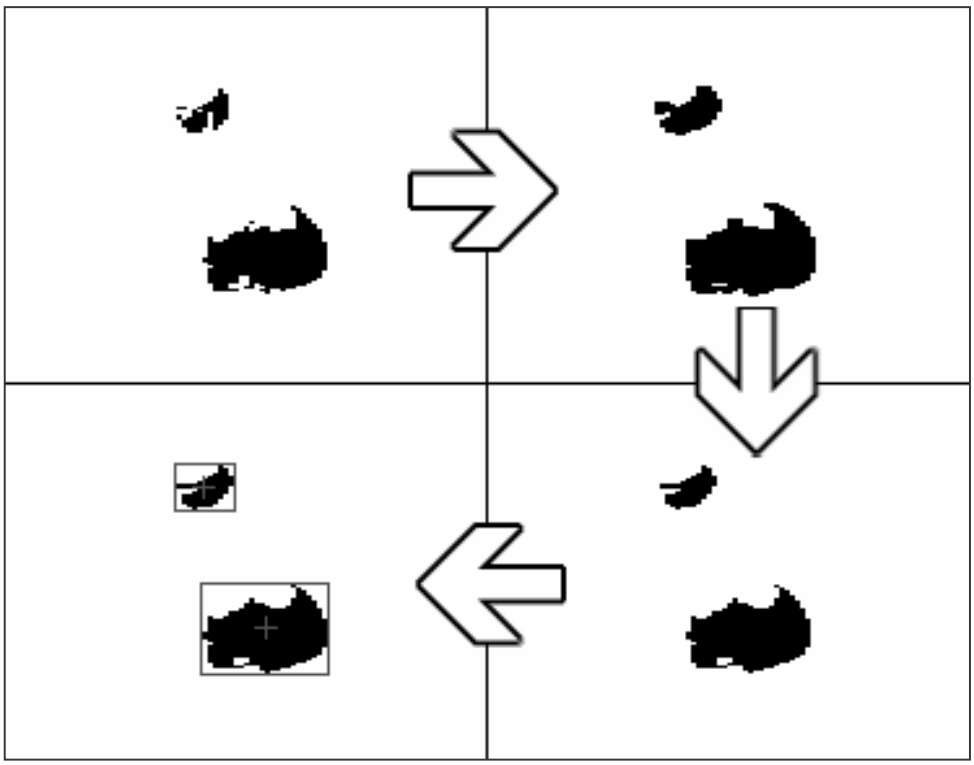
On définit les marqueurs recherchés à l'aide des paramètres suivants :

- plage U de la couleur (U_{min} et U_{max}),
- plage V de la couleur (V_{min} et V_{max}),
- surface minimale (en pixels),
- surface maximale (en pixels),
- ratio maximal entre largeur et hauteur,
- vitesse de déplacement maximale (en pixels) entre 2 images.



Le composant "Tracker" utilise un algorithme en 5 passes pour analyser les images. Le montage ci-dessous montre ces différentes passes dans le cas de la recherche de marqueurs rouges dans l'image de la page précédente :

- Dans la première passe, seuls les pixels dont les valeurs U-V sont dans les plages requises



  sont conservés.
- On applique ensuite un filtre de dilatation (3x3 pixels) dont le but est de supprimer les pixels "blancs" isolés dans des zones "noires" (typiquement dus au bruit).
- Puis on applique un filtre d'érosion (3x3 pixels) afin de supprimer les pixels "noirs" isolés dans des zones "blanches".
- À l'issue de ces trois premières étapes, l'image est prête pour la détection des marqueurs. On utilise alors un algorithme récursif qui, à partir d'un pixel noir, va parcourir tous les pixels noirs adjacents et déterminer : 1) le rectangle minimum entourant le marqueur, 2) la surface du marqueur et 3) son centre de gravité. Si la surface du marqueur ou le ratio des longueurs des côtés du rectangle ne sont pas dans la plage requise, le marqueur est éliminé.
- Connaissant la position du ou des marqueurs recherchés à l'image précédente, il faut leur faire correspondre les marqueurs "potentiels" résultants des passes 1 à 4. Pour cela, on prend simplement le plus proche marqueur potentiel de chacun des marqueurs de l'image précédente, dans la limite de la vitesse de déplacement maximale spécifiée.

Un système aussi simple a, bien entendu, des limites :

- Si l'on poursuit 2 marqueurs identiques en même temps et qu'ils se croisent, ils peuvent êtres intervertis. Toutefois, vu que l'on utilise des boules placées sur un cadre, la probabilité reste faible.
- Le composant ne peut chercher en même temps plusieurs marqueurs avec des caractéristiques différentes. Afin de détecter les boules de notre cadre, il est donc appelé 3 fois : pour les 2 boules rouges, pour la boule bleue et enfin pour la verte.
- La conséquence de la limite précédente est que le système cherche les marqueurs individuellement et non pas un agencement spatial "admissible" de marqueurs. Si l'un des vrais marqueurs sort du champ, un marqueur "parasite" peut se retrouver poursuivi à sa place. En fait, cela n'est pas un problème, car le composant "Synchronizer" ne tient pas compte des agencements spatiaux "aberrants" de marqueurs (cf. section "Synchronizer").

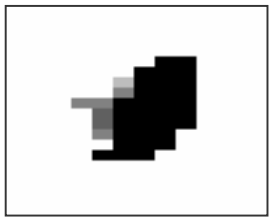
Par contre, la généralité de ce composant fait que l'on peut l'utiliser pour traquer n'importe quel ensemble de marqueurs.

Le principe utilisé pour détecter les marqueurs n'est en aucun cas adaptatif : à travers la définition des plages de couleur U-V, il est tributaire des conditions d'éclairage. Les informations de luminosité de l'image ne sont d'ailleurs pas du tout prises en compte. Nous avons bien testé un algorithme[1] où elles étaient utilisées pour corriger la dynamique U-V, mais sans grand effet.

L'influence des conditions d'éclairage apparaît nettement dans l'exemple précédent : les reflets des 2 lampes du plafond et les ombres sous les boules font que de nombreux pixels sont éliminés lors de la première passe de l'algorithme. La forme finale ressemble plus à un ovale qu'à une sphère avec le risque que son centre ne corresponde plus au vrai centre de la boule. On peut "compenser" les effets de l'éclairage en élargissant les plages U-V, mais on risque alors de traquer de faux marqueurs (comme des mains au lieu des boules rouges). C'est un compromis à trouver entre la qualité de la détection et les couleurs admissibles dans l'environnement.
Une solution envisageable serait que le filtrage selon les plages U-V se fasse en deux passages :

- un premier sur toute l'image avec des plages U-V restreintes, pour déterminer les zones où se trouvent les marqueurs sans risquer d'en trouver de faux,
- un second, limité à ces zones mais avec des plages U-V plus larges, pour détecter la plus grande partie de chacun des marqueurs.

[1] Lorsque la luminosité est très basse ou très haute, la dynamique U-V de la caméra diminue. L'idée est donc de multiplier les valeurs U-V par un gain suivant la valeur de Y. En théorie, cela permettrait d'atténuer l'effet des reflets et des ombres sur la boule.



Un autre point important est la méthode de calcul du centre des marqueurs. Initialement, nous l'assimilions au centre de leur rectangle. Cette méthode se révèle être très sensible au bruit sur le pourtour des marqueurs, car si même un unique pixel supplémentaire y apparaît, cela peut agrandir le rectangle et déplacer son centre. Surtout si l'on considère que, typiquement, un marqueur ne fait que 10 à 20 pixels de large. L'image ci-contre montre justement le bruit qui peut apparaître sur la périphérie d'un marqueur (pixels en gris).
Nous avons donc décidé de calculer leur centre de gravité. Cette méthode a l'avantage d'être nettement moins sensible au bruit, et cela a un effet de stabilisation sur la position des marqueurs. Par contre, son résultat peut être faussé par des reflets sur les boules, car ils apparaissent souvent comme des tâches "blanches" à l'intérieur du marqueur, ce qui déplace son centre de gravité.
Il n'existe donc pas de solution simple et efficace. Pour avoir le résultat le plus fiable possible, il faudrait retrouver mathématiquement la forme originale de la sphère à partir de sa "tâche" et connaître ainsi le centre exact.

Même si les marqueurs sont physiquement immobiles, à cause du bruit dans l'image, leurs centres calculés oscillent dans un rayon d'un à deux pixels. Pour les stabiliser, on effectue une pondération entre l'ancienne position (80%) et la nouvelle (20%) **uniquement** dans le cas où elles sont à moins de 5 pixels l'une de l'autre.
Comme autres techniques de stabilisation, on peut envisager l'utilisation :

- d'un histogramme des dernières positions où la position qui reviendrait le plus souvent serait considérée comme la plus exacte,
- d'un filtre de Kalman.

Il reste donc des contraintes certaines dans l'utilisation du "Tracker", mais dans le cadre de ce projet, ses possibilités conviennent tout à fait.

YUV422 Decoder

Ce composant décompresse une image encodée en YUV (4:2:2) en une image RGB (8:8:8). La conversion YUV – RGB peut-être représentée par une simple opération matricielle (les valeurs de la matrice sont standards):

$$\begin{pmatrix} r \\ g \\ b \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 1.402 & 0.5 \\ 1 & -0.3437 & -0.7143 & 0.5 \\ 1 & 1.77 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} y \\ u \\ v \\ 1 \end{pmatrix}$$

La luminance Y est codée sur un octet entre 0 et 255 tandis que les valeurs de chrominance U et V sont codées chacune sur un octet entre –128 et +127.
Les valeurs R, G, B obtenues sont codées chacune sur un octet et après décodage, elles sont donc limitées dans l'intervalle 0 – 255 (au cas où des valeurs Y, U, V incorrectes seraient passées).

Les pixels YUV étant codés deux à deux par blocs de 32 bits, chaque bloc a en fait la forme suivante : [Y_1 U, Y_2 V]. Pour convertir deux pixels adjacents en RGB, il faut donc multiplier les vecteurs suivants par la matrice de conversion :

$$\begin{pmatrix} Y_1 \\ U \\ V \\ 1 \end{pmatrix} \quad et \quad \begin{pmatrix} Y_2 \\ U \\ V \\ 1 \end{pmatrix}$$

La résolution horizontale de la chrominance est bien deux fois moindre que celle de la luminance.

Nous avons également développé un logiciel prototype qui utilise la puce graphique de l'ordinateur pour faire la conversion YUV(4:2:2) -> RGB en hardware et afficher directement le résultat à l'écran, agrandi et lissé (par exemple la vidéo 320x240 est affichée en 640x480 plein écran). Bien que le résultat obtenu soit extrêmement fluide, nous avons finalement décidé de ne pas utiliser cette méthode dans le composant final, car:

- elle utilise l'API Rave qui est spécifique à MacOS et n'est donc pas portable,
- elle n'est pas utilisable en parallèle avec l'API OpenGL qui est utilisée pour le rendu 3D.

OBJ Importer

Le format de fichier WaveFront OBJ est un format texte universel pour stocker la géométrie d'objets 3D (ces fichiers ne contiennent pas d'images de textures ou d'animations). Nous avons choisi ce format car il est simple à décoder et quasiment tous les logiciels de création 3D le supportent. On peut également éditer les fichiers OBJ avec un simple éditeur de texte.

À titre d'exemple, voici un cube de 200 unités de côté décrit au format OBJ :

```
v -100 -100 100
v -100 100 100
v 100 -100 100
v 100 100 100
v 100 -100 -100
v 100 100 -100
v -100 -100 -100
v -100 100 -100

f 1 2 4 3
f 3 4 6 5
f 5 6 8 7
f 7 8 2 1
f 2 8 6 4
f 7 1 3 5
```

Un fichier OBJ est donc une liste d'éléments (un par ligne) dont les principaux sont :

- Point: **v** x y z (où x, y et z sont des coordonnées dans l'espace),
- Texture: **vt** u v (où u et v sont des coordonnées sur la texture de l'objet),
- Normale: **vn** x y z (où x, y et z décrivent un vecteur 3D),
- Face: **f** $p_1/t_1/n_1$ $p_2/t_2/n_2$… $p_n/t_n/n_n$ (où p_i, t_i et n_i sont les indices des éléments points, textures et normales qui forment la face avec $n \geq 3$).

Les éléments de texture et normales sont facultatifs.

Le composant doit donc interpréter les données textes OBJ (obtenues depuis un fichier OBJ par le composant "File Reader") et les convertir dans un format binaire optimisé pour le composant "Renderer", lequel dessinera les objets 3D à l'écran. Au passage, les faces de plus de 3 côtés sont décomposées en triangles.

TGA Importer

Le format de fichier Targa TGA permet de stocker des images de façon très simple. Le fichier est composé d'un en-tête succinct qui décrit les caractéristiques de l'image (dimensions, nombres de couleurs, table des couleurs…), suivi des données de l'image même. Là aussi, la majorité des logiciels graphiques permettent d'enregistrer des images dans ce format.

Voici le format des en-têtes de fichier TGA :

```
unsigned char       identificationFieldSize (in bytes)
unsigned char       hasColorMap (0 if there is none)
unsigned char       format (2 if RGB)

short               colorMapOrigin (*)
```

short	colorMapLength (*)
unsigned char	colorMapEntrySize (*)
short	imageOriginX (*)
short	imageOriginY (*)
short	imageWidth (in pixels)
short	imageHeight (in pixels)
unsigned char	imagePixelSize (bits per pixel)
unsigned char	imageDescriptor (*)

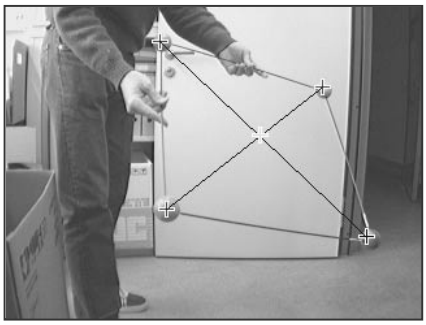
(*) Ces champs ne sont pas utilisés par le composant car il ne lit que les fichiers TGA non compressés, en milliers ou millions de couleurs.

Le composant va convertir les données TGA (chargées depuis un fichier TGA par le composant "File Reader") en une image, qui sera utilisable comme texture d'objet par le composant "Renderer".

Synchronizer

Ce composant est le plus important de l'ensemble : c'est à lui que revient la charge de synchroniser les images virtuelles sur le monde réel et générer ainsi la réalité augmentée. Il dispose pour cela de deux sources d'information : le cadre de calibration et le capteur InterTrax 2.

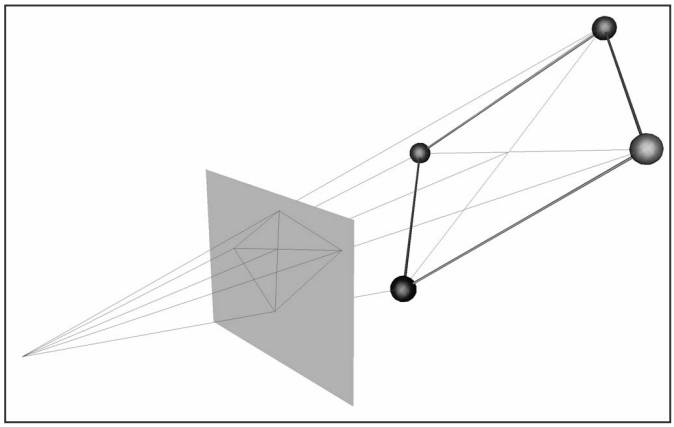
Le composant commence par déterminer si le cadre est visible[1] dans les images de la caméra grâce aux informations retournées par le composant "Tracker". La réponse est positive si les deux marqueurs rouges, le marqueur bleu et le vert sont correctement traqués. L'ambiguïté due aux deux boules rouges est résolue en calculant l'intersection des deux diagonales et en s'assurant qu'elle se trouve bien à l'intérieur du cadre. À la fin de cette étape, on connaît donc la position de chacune des boules à l'écran, c'est-à-dire dans le plan de projection de la caméra.

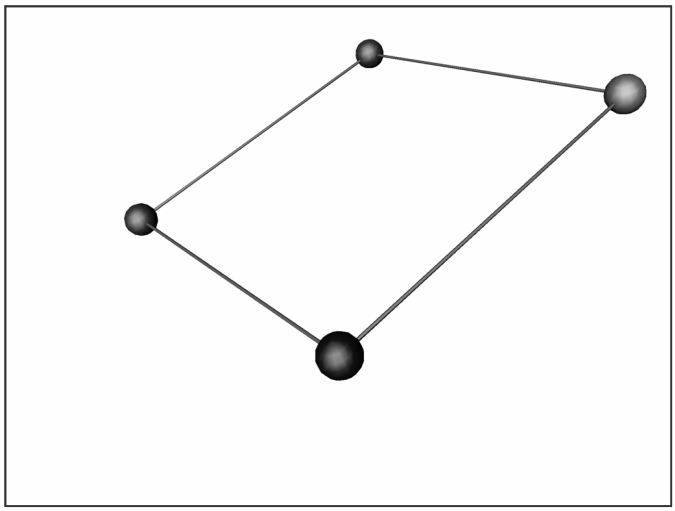


Toujours dans l'hypothèse où le cadre serait visible, l'étape suivante est de calculer sa position et son orientation dans l'espace de la caméra.
Nous faisons deux hypothèses :

- nous assimilons l'iBot à une caméra de type pinhole,
- nous ne tenons pas compte de la distorsion optique[2].

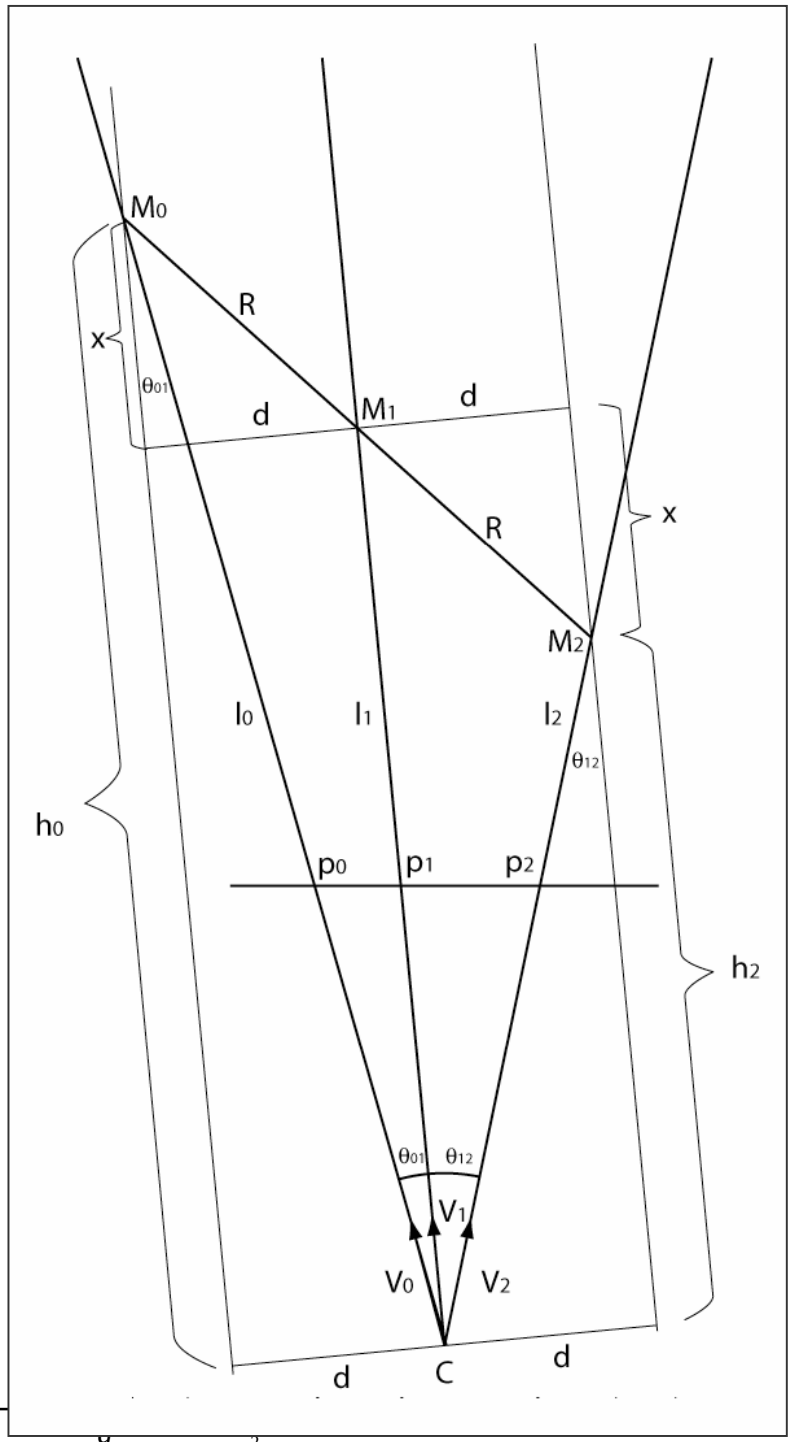
Dans le respect des hypothèses précédentes, les deux images ci-contre montrent la projection du cadre sur le plan focal de la caméra et l'image du cadre dans ce même plan focal.





[1] Par "visible", on entend visible pour le système de réalité virtuelle, car il se peut que le cadre soit bien dans le champ de la caméra mais ne soit pas détecté.
[2] En effet, pour vraiment la prendre en compte dans tout le système, il faudrait que le composant "Renderer" puisse calculer les images virtuelles avec cette même distorsion. Mais la technologie utilisée (OpenGL) ne le permet pas, à moins d'utiliser des approches complexes qui peuvent être grandes consommatrices de temps machine.

Plaçons-nous dans le plan formé par l'une des diagonales du cadre et le point focal. Les centres des deux boules sont M_0 et M_2. Le centre de la diagonale est en M_1. Les projections de ces trois points dans le plan image sont p_0, p_1 et p_2, C représentant le centre focal.
Les coordonnées à l'écran de p_0 et p_2 sont obtenues depuis le composant "Tracker". Celles de p_1 correspondent à l'intersection des deux diagonales du cadre dans le plan focal. La coordonnée de profondeur de p_0, p_1 et p_2 est la focale de la caméra (en pixels).



On cherche donc les coordonnées 3D de M_0 et M_2 dans le référentiel de la caméra. Or nous avons :

$$M_0 = l_0 \cdot \vec{V}_0 \; où \; \vec{V}_0 = \frac{\overrightarrow{Cp_0}}{\left\|\overrightarrow{Cp_0}\right\|}$$

$$M_2 = l_2 \cdot \vec{V}_2 \; où \; \vec{V}_2 = \frac{\overrightarrow{Cp_2}}{\left\|\overrightarrow{Cp_2}\right\|}$$

On obtient l_0 et l_2 par :

$$l_0 = \frac{d}{\sin(\theta_{01})} \; et \; l_2 = \frac{d}{\sin(\theta_{12})}$$

$$où \begin{cases} \theta_{01} = \cos^{-1}(\vec{V}_0 \cdot \vec{V}_1) \\ \theta_{12} = \cos^{-1}(\vec{V}_1 \cdot \vec{V}_2) \end{cases}$$

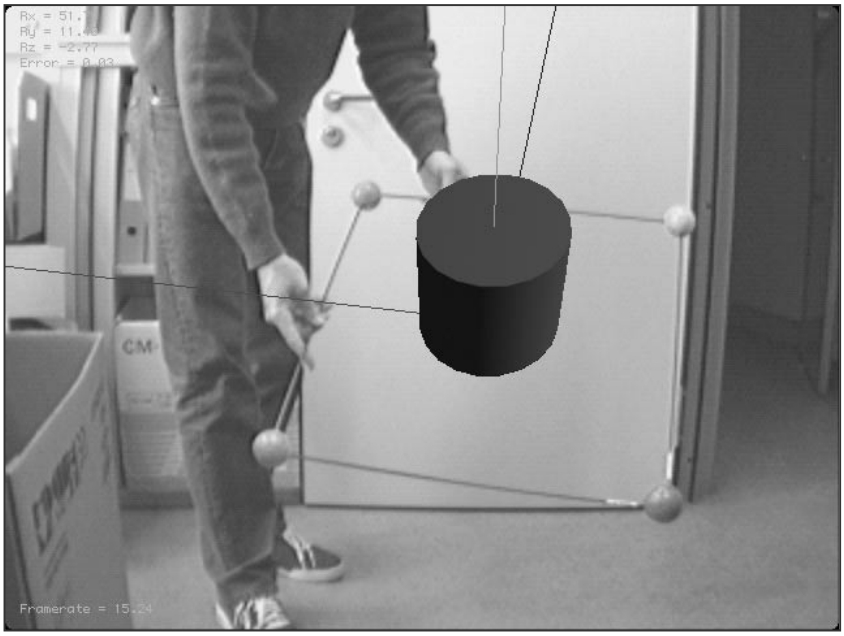
Enfin les relations trigonométriques nous permettent d'écrire :

$$R = \frac{1}{2}\sqrt{\frac{d^2}{\tan(\theta_{12})^2} - \frac{2 \cdot d^2}{\tan(\theta_{12}) \cdot \tan(\theta_{01})} + \frac{d}{\tan(\theta_{01})^2} + 4 \cdot d^2}$$

Soit (avec 2R la longueur connue de la diagonale) :

$$d = \frac{2 \cdot R}{\sqrt{\frac{1}{\tan(\theta_{12})^2} - \frac{2}{\tan(\theta_{12}) \cdot \tan(\theta_{01})} + \frac{1}{\tan(\theta_{01})^2} + 4}}$$

L'application de cette méthode de résolution sur les deux diagonales permet de connaître les positions des coins du cadre dans l'espace de la caméra. Deux côtés adjacents (bleu-rouge et bleu-vert) sont utilisés pour définir les axes X et Z d'un référentiel tandis que l'axe Y (sortant du plan[1]) est construit avec un simple produit vectoriel. Ce référentiel établit un "pont" entre les mondes virtuels et réels, et l'on peut alors ajouter des objets virtuels au monde réel comme le montre l'image ci-dessous (les axes X, Y et Z sont représentés en rouge, vert et bleu) :
On note d'ailleurs que cette méthode fonctionne pour des cadres rectangulaires ou carrés.



Par contre, si les informations passées sont erronées (mauvais marqueurs traqués par exemple), la méthode va retourner des résultats aberrants. Le composant va donc s'assurer en premier lieu que les quatre marqueurs forment bien un carré, du moins dans des tolérances données : les centres des diagonales ne doivent pas êtres distants de plus de 4 cm et les diagonales doivent former un angle droit à 10° près[2]. Le référentiel construit est également corrigé pour être parfaitement orthonormal, afin que les objets virtuels ne soient pas déformés.
Malgré la stabilisation des marqueurs (cf. composant "Tracker"), le référentiel peut vibrer légèrement. Même si cela n'est pas gênant pour les démonstrations envisagées, dans l'idéal, on devrait également stabiliser le référentiel, avec un filtre de Kalman par exemple.

[1] L'axe vertical est l'axe Y et non l'axe Z pour respecter les conventions OpenGL.
[2] Il n'est pas nécessaire de contrôler la longueur des diagonales car la méthode de résolution de la page précédente implique qu'elles soient exactes.

Comme dit au début de cette section, nous n'avons finalement pas tenu compte de la distorsion optique pour des raisons techniques. A priori, les résultats du "Synchronizer" n'ont donc pas de sens. En fait, la distorsion reste négligeable dans une bonne partie de l'image, et surtout, elle ne devrait pas influencer les résultats du "Synchronizer" : si le cadre se trouve dans un coin de l'image, là où la distorsion est importante, sa projection sera déformée ; en toute logique, le calcul de la position du cadre dans l'espace à partir de cette projection déformée donnera alors une solution qui ne sera pas un cadre (les points ne seront pas coplanaires par exemple), et cette solution sera rejetée.
Toutefois, si l'on avait voulu tenir compte de la distorsion, il aurait fallu procéder comme suit :

- Une fois les marqueurs détectés, corriger leurs positions à l'écran en appliquant la distorsion inverse (grâce à une look-up-table par exemple),
- Calculer la position du cadre dans l'espace de la caméra avec la méthode habituelle,
- Effectuer le rendu 3D des objets virtuels dans un buffer distinct (avec couche alpha), appliquer la distorsion sur l'image du buffer, puis superposer le tout sur l'image vidéo. Pour appliquer la distorsion, on peut soit déformer l'image pixel par pixel, soit l'utiliser comme texture sur une surface 3D qui reproduirait l'effet de la distorsion.

La deuxième tâche du composant "Synchronizer" est de fusionner les informations obtenues depuis le cadre avec celles du capteur InterTrax 2, retournées par le composant "Data Converter". Il existe en fait deux cas possibles :

- Le cadre est visible et le référentiel a pu être construit : il y a redondance d'information (données de position et d'orientation avec le référentiel, données d'orientation avec le capteur) mais aucune n'est 100% fiable. On utilise typiquement un filtre de Kalman pour traiter ces données et effectuer la meilleure calibration possible.
- Seules les informations de l'InterTrax 2 sont disponibles : on les utilise alors pour mesurer la différence angulaire entre la dernière calibration et l'orientation actuelle, ce qui permet de construire une matrice de rotation que l'on applique au monde virtuel.

Lors de l'implémentation de cette partie du "Synchronizer", pour une raison inconnue, l'InterTrax 2 a commencé à présenter une très forte dérive / hystérésis : si on le tourne puis on le fait revenir à sa position initiale, il peut y avoir jusqu'à des dizaines de degrés d'offset. Dans le laps de temps restant, il n'a pas été possible d'en déterminer la cause ou d'y trouver un remède.

La fusion entre les informations de l'InterTrax 2 et celles des marqueurs n'est donc pas fonctionnelle. Cela dit, les quelques essais effectués nous ont fait réaliser l'importance d'avoir des latences identiques dans ces deux flux d'information. En effet, l'InterTrax 2 a une latence d'environ 10 ms contre 50 ms pour l'iBot. Lors d'un mouvement de tête, les images virtuelles se retrouvent donc "en avance" sur les images réelles : il est nécessaire de retarder les données de l'InterTrax 2.

Rotoscoper

Le rôle de ce composant basé sur l'API OpenGL est d'afficher l'image venant de la caméra "en fond" pour que le composant "Renderer" puisse ensuite dessiner par-dessus les objets 3D. Il existe deux approches :

- soit copier l'image vidéo directement dans le framebuffer[1],
- soit créer une texture OpenGL qui contient l'image vidéo mise à jour en permanence, puis dessiner un rectangle avec cette texture dans le framebuffer.

Même si elle paraît plus complexe, la deuxième approche est optimale. En effet, elle rend indépendante la taille de la vidéo de la taille de rendu : si la vidéo est en 320x240 et l'écran en 800x600, OpenGL se chargera d'étirer la texture pour qu'elle remplisse tout l'écran. En outre, cet agrandissement est effectué en hardware; il est donc lissé et très rapide.

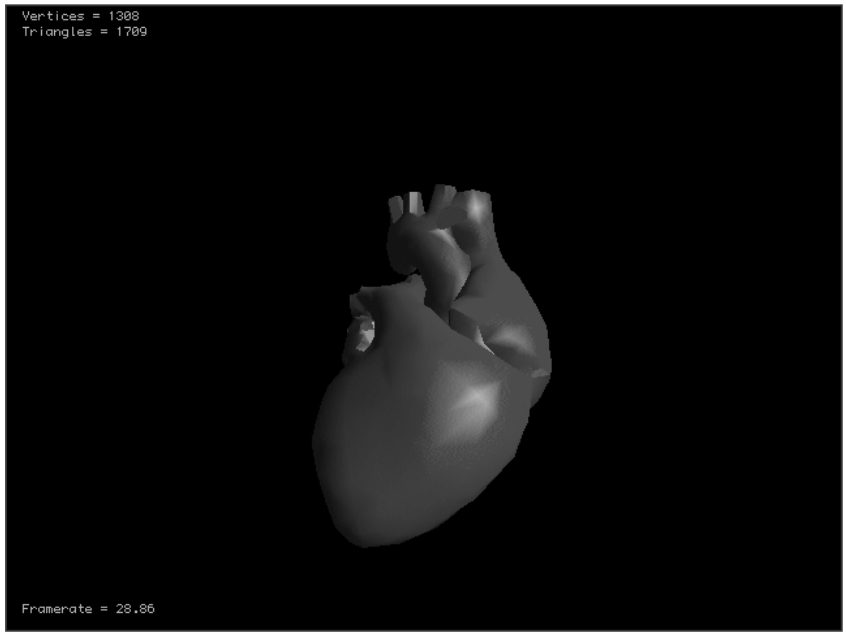
Il reste cependant une petite contrainte dans la mesure où les textures doivent avoir des dimensions en puissance de 2, ce qui n'est pas le cas des images vidéos : une image en 320x240 occupera une texture en 512x256, une partie restant inoccupée. Heureusement, la grande quantité de mémoire vidéo disponible sur les ordinateurs actuels rend cette contrainte négligeable.

À chaque fois qu'une nouvelle image vidéo est capturée, le composant "Rotoscoper" utilise donc le composant "YUV422 Decoder" pour la décompresser dans une texture RGB OpenGL, qui est ensuite dessiné dans un rectangle qui occupe tout l'écran.

Renderer

[1] Le framebuffer est un buffer vidéo dans lequel OpenGL effectue le rendu avant de copier son contenu à l'écran. Si OpenGL dessinait directement à l'écran, on pourrait voir le dessin s'effacer puis se redessiner morceau par morceau en permanence, ce qui est très pénible visuellement.

Ce composant utilise OpenGL pour dessiner un objet 3D (importé par le composant "OBJ Importer") avec sa texture (elle-même importée par le composant "TGA Importer") aux position et orientation spécifiées par le composant "Synchronizer".



Le "Renderer" implémente également des possibilités d'animation basiques : disposant de deux objets 3D similaires avec leurs propres positions et orientations, il peut interpoler l'objet intermédiaire (par exemple à 25% du premier et 75% du second) et le dessiner. Par exemple, ce système permet d'animer facilement un cœur si l'on a un objet 3D pour la systole et un autre pour la diastole.

Pour donner plus de relief aux objets 3D, il est indispensable de leur ajouter des ombres et des reflets. Le "Renderer" utilise donc une source d'éclairage omnidirectionnelle unique placée arbitrairement au nord-est, au-dessus de l'utilisateur.
Bien entendu ce système est très sommaire : pour plus de réalisme, il aurait fallu placer une ou plusieurs sources d'éclairage avec les mêmes emplacements et caractéristiques que celles du monde réel. Toutefois, comme nous le verrons dans le prochain chapitre, cela n'aurait pas eu de sens dans le cadre des démonstrations envisagées pour ce prototype.

Par ailleurs, sans entrer dans les détails techniques (se référer au code source du programme pour plus d'information), ce composant a été optimisé pour utiliser compiled arrays d'OpenGL, ce qui permet un gain d'environ 10% sur les performances de rendu.

EXEMPLES D'APPLICATIONS

Les applications du prototype de ce projet ont pour but de montrer le potentiel de la réalité augmentée dans des domaines types. Il ne faut pas oublier que le système développé est un prototype, et en aucun un produit "prêt à consommer" : les exemples d'applications ne donnent donc qu'une idée grossière de ce que pourrait être le produit final.

Affichage d'organes sur un patient

L'idée est d'afficher directement les informations obtenues depuis les appareils d'observation médicaux (scanner, radio…) sur le patient lors de l'établissement d'un diagnostic ou d'une intervention chirurgicale. Bien entendu, cela n'améliora pas à priori la qualité des résultats, mais par contre, cela devrait améliorer grandement le "confort" de travail du praticien : visualiser directement les organes à l'intérieur du patient est nettement plus pratique que de regarder un écran. De plus cela permettrait au chirurgien d'intervenir très précisément et directement à l'endroit souhaité, sans risque d'erreur. Le chirurgien aurait également accès en permanence aux données physiologiques (pouls, pression sanguine…) sur ses lunettes 3D.

Affichage de câblage électrique et tuyauterie dans un mur

Cette démonstration mets en valeur l'intérêt de la réalité augmentée si l'on doit accéder à des conduits et des câbles électriques emmurés, pour effectuer des travaux de réparation, alors qu'aucun accès n'est prévu. On devrait normalement détruire toute la surface du mur, mais c'est particulièrement inefficace, voire dangereux, si l'on n'est pas sûr de ce qui se trouve dans le mur et où (comme des câbles électriques sous-tension, ou des conduites de gaz).

Dans l'hypothèse où les plans seraient disponibles, ils seraient affichés virtuellement sur le mur. Dans le cas contraire, on peut imaginer des systèmes de capteurs (inductifs, capacitifs…) qui détermineraient l'agencement des éléments dans le mur. Par-dessus les tracés des conduits, on superposerait ensuite des informations indiquant dans quels sens circulent les fluides, quels câbles sont alimentés et sous quelle tension, quelles sont leurs destinations…

Un tel système permettrait sans aucun doute à l'ouvrier de travailler nettement plus vite, plus efficacement et dans de meilleures conditions de sécurité.

CONCLUSION

Même si le résultat final n'atteint pas celui escompté au départ à cause de problèmes indépendants de notre volonté, ce projet a permis de prendre conscience du potentiel de la réalité augmentée et des problèmes de réalisation qu'elle pose.

Les aspects technologiques (puissance des ordinateurs, qualité des images, lunettes 3D…) n'étant plus le facteur limitant aujourd'hui, le principal problème est sans aucun doute la synchronisation du monde virtuel avec le monde réel. Il est en effet extrêmement difficile de construire un système de synchronisation qui ne nécessite pas de modifier l'environnement, qui soit stable, qui ne dérive pas, et enfin, qui a une faible latence. La solution adoptée ici (cadre de boules de couleurs et capteur d'orientation) a des limites sérieuses et peut difficilement être envisagée dans une solution commerciale et professionnelle, mais suffit pour tester les concepts de la réalité augmentée.
À court terme, la synchronisation nécessitera encore probablement la pose de marqueurs dans l'environnement avec toutes les restrictions et imprécisions que cela implique. À long terme, l'idéal serait de disposer d'un système de positionnement GPS ultra précis couplé à un capteur d'orientation, ce qui permettrait de disposer en permanence d'informations absolues et fiables de position et d'orientation. Un tel système pourrait par contre ne pas fonctionner correctement en intérieur, à moins d'installer un GPS local.

Les applications pour la réalité augmentée sont énormes, et même si aujourd'hui elle est encore réservée aux domaines pointus (militaire, médical), les rapides développements des périphériques de réalité virtuelle ces dernières années et la chute de leurs prix laissent présager une arrivée dans le domaine du grand public dans un avenir proche.

Des travaux futurs dans la lignée de ce projet pourraient comprendre l'analyse et la correction des défauts du capteur InterTrax 2, une détection des marqueurs adaptative suivant les conditions d'éclairage, et enfin l'amélioration de la qualité d'image (640x480 au lieu de 320x240).

Je tiens à remercier M. Clavel, Nicolas Chauvin et M. Baur pour avoir mis à ma disposition le matériel requis pour le projet et m'avoir permis de travailler pendant un semestre dans l'ambiance sympathique du VRAI Group. Je remercie également tous les membres du VRAI Group et de 2C3D pour leur aide dans la réalisation du projet.

BIBLIOGRAPHIE

[1] Eric M. Foxlin, US Patent #5,645,077
Per Krogh Hansen, Vladimir Kogan, US Patent 5,953,683

[2] Site web Human Interface Device pour les périphériques USB :
http://www.usb.org/developers/hidpage.html

[3] Site web des produits Texas Instruments :
http://focus.ti.com/docs/prod/productfolder.jhtml?genericPartNumber=TSB15LV01

[4] Jean-Yves Bouguet, "*Complete Camera Calibration Toolbox for Matlab*"
http://www.vision.caltech.edu/bouguetj/