

Alexandre Beaumont
Thomas Brunel
David Trias

Projet INVIR

Ajout d'information en réalité
augmentée sur un modèle 3D
texturé sans marqueurs

Février - Avril 2013

Contexte 4

Problème 5

Définition et installation des outils	5
Diagramme pieuvre	6

Veille technologique 7

OpenCV	7
PTAM	7
<i>libCVD</i>	7
<i>TooN</i>	8
<i>GVars3</i>	8
<i>LAPACK</i>	8
<i>BLAS</i>	8
KLT	9
OpenGL	9
SOIL	9
HandyAR	9
<i>Exemple d'utilisation du projet HandyAR</i>	10

Démarche 11

Installation des outils	11
<i>Sous Windows</i>	11
<i>Sous GNU/Linux</i>	11
<i>Sous Mac OS X</i>	12
Découverte et prise en main des outils	12
Ajout d'informations	12
<i>Exemple d'un cube modélisé avec PTAM</i>	13
<i>Exemple d'un cube avec une texture</i>	13
<i>Exemple d'un cube utilisant la texture du bâtiment des carènes</i>	14
<i>Exemple de la résidence n°2 avec une erreur dans l'emplacement des faces</i>	14
<i>La résidence n°2 complètement modélisée</i>	15
<i>Les trois bâtiments modélisés avec leurs textures</i>	15
Tests grandeur nature et corrections	16
<i>Ajout des bâtiments sans réglages</i>	16
<i>Réglage des bâtiments en cours</i>	17

<i>Ajout des bâtiments une fois les réglages terminés (1)</i>	17
<i>Ajout des bâtiments une fois les réglages terminés (2)</i>	18
Solution adoptée	19
Scénario d'utilisation	19
Bilan critique	20
Idées d'amélioration :	20
<i>La gestion des occlusions entre le réel et le virtuel</i>	20
<i>Exemple d'occlusion de la réalité dans le monde virtuel</i>	21
<i>Amélioration de la stabilité</i>	21
Pour voir plus loin	21
<i>Ajout d'une ambiance sonore</i>	21
<i>Schématisation de l'évolution au cours du temps</i>	21
Bibliographie	22
Annexes	23
Annexe 1 : Installation de PTAM sous Ubuntu 12.10	23
Annexe 2 : Exemples de code	24

Contexte

Dans l'optique de pouvoir afficher différentes informations sur une maquette de site historique, il nous a été demandé de développer un programme multi-plateformes permettant d'ajouter, en réalité augmentée, les informations manquantes. L'ajout de ces informations ne doit pas dépendre de marqueurs susceptibles de dénaturer l'environnement. La réalité augmentée utilisée ne se base donc pas sur des marqueurs, mais sur l'analyse d'un flux vidéo.

Pour cela, nous nous sommes basés sur la maquette de l'Ecole Centrale de Nantes visible au Hall A, sur laquelle nous sommes venus rajouter quelques éléments manquants (bâtiments existants et à venir). Il est également important de prendre en compte la texture, dans le cas d'ajout de nouveaux bâtiments. Nous avions à notre disposition une webcam et un ordinateur, le but étant de filmer la scène avec la webcam pour visualiser l'ajout d'information sur l'écran de l'ordinateur.

Problème

Ce projet comporte plusieurs étapes importantes à franchir afin d'être mené à bien.

Définition et installation des outils

Les outils informatiques dont nous aurons besoin doivent :

- Déetecter et suivre des points dans l'espace à partir d'une analyse d'un flux vidéo.
- Pouvoir positionner la caméra à partir de ces points.
- Nous permettre de modéliser les bâtiments informatiquement, en permettant l'ajout de texture, puis de le afficher à l'écran.
- Etre compatibles entre-eux : si nous avons un logiciel qui s'occupe de l'analyse de la vidéo, et un autre qui modélise les objets, ils doivent pouvoir communiquer entre-eux.
- Etre multi-plateforme.

Pour les outils matériels, le projet nécessite une webcam USB ou firewire et un ordinateur portable.

Travail à réaliser

Le projet commence par la réalisation d'un état de l'art des différents outils libres à disposition, et du choix d'un ou plusieurs de ces outils.

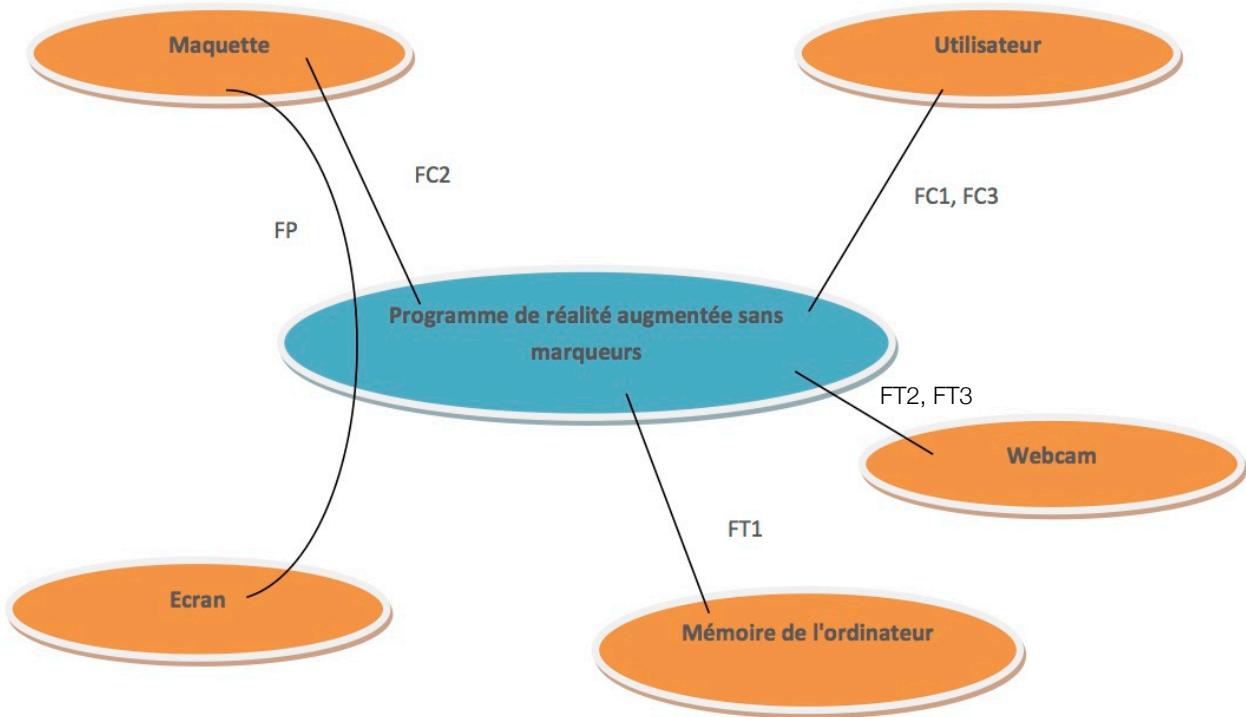
Il conviendra ensuite d'installer ces outils sur les différents OS, de découvrir leurs fonctionnalités, et de les essayer sur des cas simples. Il faudra auparavant avoir installé Linux sur les ordinateurs.

Une fois familiarisé avec les logiciels choisis, nous pourrons écrire ou adapter les programmes nécessaire au projet, puis commencer la modélisation. Pour les textures des bâtiments, nous prendrons des photos de ces derniers.

Chaque semaine, nous rendrons compte de notre avancement et de nos difficultés rencontrées, et nous proposerons une mise à jour du planning du projet.

Diagramme pieuvre

Voici le diagramme pieuvre du projet :



- FP : Ajouter de la réalité augmentée sans marqueur sur la maquette de l'ECN : observer à l'écran les nouveaux bâtiments par dessus la maquette.
- FC1 : le programme créé doit être multiplateforme (Mac OS, Windows, Linux).
- FC2 : Eviter les reflets de la vitre de la maquette
- FC3 : Suivre les mouvements de l'utilisateur
- FT1 : Récupérer les positions, dimensions et les textures des nouveaux bâtiments
- FT2 : Récupérer le flux vidéo
- FT3 : Positionner la webcam et récupérer les points d'intérêt de la maquette

Veille technologique

Le domaine d'ajout d'informations en réalité augmentée sur une scène comporte un très grand nombre d'outils. Cependant, la difficulté principale de ce projet était d'utiliser de la réalité augmentée sans marqueur, ce qui signifie que le programme doit pouvoir directement trouver les points d'intérêt de la scène et de calculer l'emplacement des informations en fonction de ces derniers. Cela nous a donc permis de limiter notre recherche. Voici la liste des différents outils que nous avons étudiés :

OpenCV

OpenCV (Open Computer Vision) est une bibliothèque comportant plusieurs interfaces (C, C++, Java) et prend en charge Windows, Linux, Mac OS, iOS et Android. Cette bibliothèque a été conçue pour être efficace en terme de calculs et avec un fort accent mis sur les applications en temps réel. Développée en C/C++ optimisé, elle peut prendre avantage des processeurs multi-coeurs. Les plages 'utilisation de cette bibliothèque vont de l'art interactif à l'inspection de mines en passant par la robotique avancée.



PTAM

PTAM est un système de suivi de caméra pour réalité augmentée ne nécessitant ni marqueurs, ni cartes, ni modèle connus et ni capteurs inertIELS. Afin de fonctionner correctement, PTAM nécessite l'installation préalable de plusieurs bibliothèques dont voici les principales :

libCvD

libCvD est une bibliothèque performante et portable développée en C++ pour la vision par ordinateur, et le traitement d'image et de vidéo. Cette bibliothèque est conçue de telle manière que chaque partie peut être utilisée indépendamment des autres, au cas où la totalité ne serait pas nécessaire. Le module d'acquisition vidéo fournit une interface simple et uniforme pour les vidéos provenant d'une grande variété de sources (en direct ou enregistrées) et permet un accès facile aux données des pixels. De même, le module de chargement / sauvegarde d'image fournit des interfaces simples et uniformes pour charger et enregistrer des images à partir de bitmaps vers des images de 64 bits par canal RGBA.

TooN

TooN est une bibliothèque numérique implémentée en C++ et qui est conçue pour fonctionner efficacement sur un grand nombre de petites matrices. De plus elle offre un accès facile à un grand nombre d'algorithmes incluant la décomposition et l'optimisation de matrices.

TooN a été conçue de manière à s'intégrer parfaitement avec libCVD et GVars3.

GVars3

Gvars3 est une partie de libCVD.

LAPACK

LAPACK (Linear Algebra PACKage) est un outil codé en Fortran 90 et fournissant des routines afin de résoudre des systèmes d'équations linéaires, de résoudre des systèmes d'équations linéaires aux moindres carrés, des problèmes aux valeurs propres, et des problèmes de valeurs singulières. Les factorisations de matrices associées (LU, Cholesky, QR, SVD, Schur, Schur généralisé) sont également prévues, de même que les calculs connexes tels que la réorganisation des factorisations de Schur et les numéros de situation d'estimation. Les matrices denses sont traitées, mais pas les matrices creuses. Chaque fonction est prévue pour des matrices complexes ou réelles, en simple ou double précision.

BLAS

Basic Linear Algebra Subprograms (BLAS) sont un ensemble de fonctions standardisées (interface de programmation) réalisant des opérations de base de l'algèbre linéaire comme des additions de vecteurs ou des multiplications de matrices.

Ces fonctions ont d'abord été publiées en 1979 et sont utilisées dans des bibliothèques plus développées comme LAPACK. Largement utilisées pour le calcul haute performance, ces fonctions ont été développées de manière très optimisée par des constructeurs de calculateurs comme Intel et AMD, ou encore par d'autres auteurs (Goto ([en](#)) BLAS et ATLAS ([en](#)) - une version portable de BLAS - en sont des exemples). Les tests de performance LINPACK utilisent massivement la fonction multiplication de matrices générales (DGEMM) de BLAS.

KLT

KLT est une mise en œuvre, en langage C, d'un tracker de fonction pour la vision par ordinateur. Le code source est dans le domaine public. Il permet de définir les points d'intérêts sur une image ou une vidéo. Il nous a permis, entre autre, de pouvoir vérifier le bon fonctionnement d'OpenCV puisqu'il utilise cette bibliothèque.

OpenGL

OpenGL (**O**pen **G**raphics **L**ibrary) est une spécification qui définit une API multiplate-forme pour la conception d'applications générant des images 3D (mais également 2D). Elle utilise en interne les représentations de la géométrie projective pour éviter toute situation faisant intervenir des infinis.



L'interface regroupe environ 250 fonctions différentes qui peuvent être utilisées pour afficher des scènes tridimensionnelles complexes à partir de simples primitives géométriques. Du fait de son ouverture, de sa souplesse d'utilisation et de sa disponibilité sur toutes les plates-formes, elle est utilisée par la majorité des applications scientifiques, industrielles ou artistiques 3D et certaines applications 2D vectorielles. Cette bibliothèque est également utilisée dans l'industrie du jeu vidéo où elle est souvent en rivalité avec la bibliothèque de Microsoft : Direct3D. Une version nommée OpenGL ES a été conçue spécifiquement pour les applications embarquées (téléphones portables, agenda de poche, consoles de jeux...).

SOIL

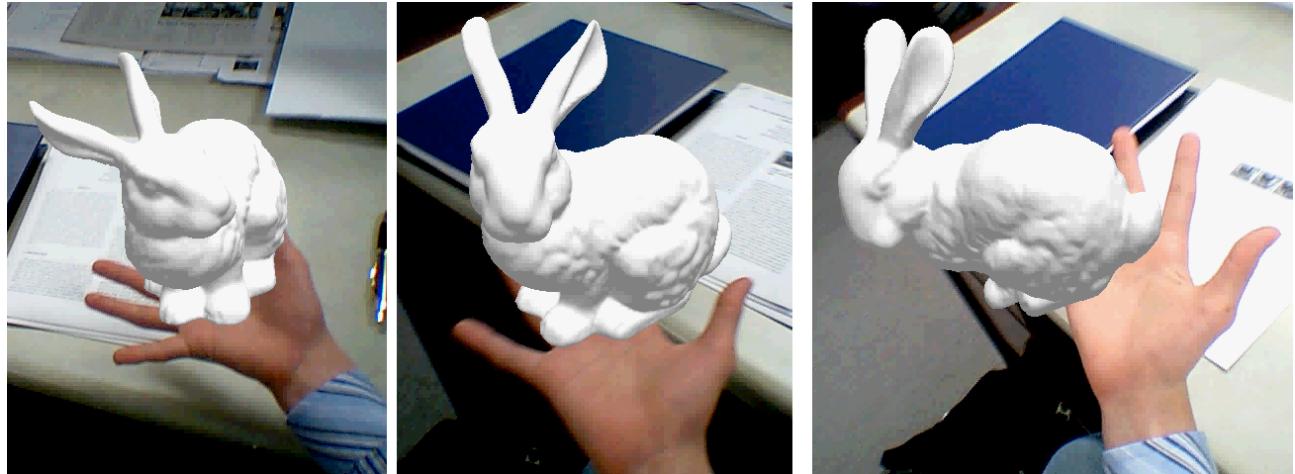
SOIL (**S**imple **O**pen**G**raphics **I**mage **L**ibrary) est une bibliothèque en C utilisée pour le chargement de textures dans OpenGL. SOIL peut aussi être utilisé pour sauvegarder ou charger des images sous de nombreux formats (BMP, PNG, JPG, TGA, DDS, PSD, HDR, ...)

HandyAR

L'un des projets en réalité augmentée sans marqueurs les plus aboutis à l'heure actuelle est le projet HandyAR, développé à l'Université de Californie et permettant d'utiliser la main de l'utilisateur en guise de marqueur, en se basant sur un algorithme de reconnaissance des pointes des doigts.

Le Handy AR présente une interface utilisateur basée sur la vision qui suit la main tendue de l'utilisateur pour s'en servir comme modèle de référence pour la réalité augmentée. Une main

modèle est réalisé dans une étape de calibrage par la mesure de la position de chaque bout des doigts par rapport aux autres.



Exemple d'utilisation du projet HandyAR

Démarche

Tout d'abord, il a fallu définir quels outils nous serons nécessaires, puis installer ces derniers sur les trois différents systèmes d'exploitation ciblés, à savoir GNU/Linux, Windows et Mac OS X (les raisons du choix de PTAM sont précisées plus loin).

Installation des outils

Sous Windows

L'installation d'OpenCV n'a pu être menée à bout car cela demandait beaucoup trop de temps et restait très compliqué. La liaison des libraires avec l'IDE (Code::Blocks ou Visual Studio) soulevait de erreurs, à de rares exemples d'utilisation près. C'est à partir de ce moment que nous avons décidé d'abandonner le projet sous Windows. PTAM n'est d'ailleurs pas originellement destiné à fonctionner sous Windows.

Sous GNU/Linux

L'installation d'OpenCV a été compliquée et relativement chronophage.

L'installation de PTAM demande au préalable d'installer plusieurs bibliothèques plus ou moins volumineuses (libCVD, TooN, GVars3, LAPACK, BLAS...). Cependant, l'installation de ces bibliothèques a nécessité une grande charge de travail et beaucoup de temps car il a fallu installer les bonnes versions de chaque et s'assurer que par exemple, la version de libCVD choisie était compatible avec la version de LAPACK choisie... Par ailleurs, il est précisé, dans les prérequis de PTAM, que l'installation de ce dernier ne fonctionnera que pour un ordinateur possédant une carte graphique NVidia (ce qui explique pourquoi nous n'avons pu installer PTAM que sur un seul des deux ordinateurs en notre possession et tournant sous Linux). Finalement, l'installation de PTAM sous GNU/Linux a été une réussite et nous avons alors pu commencer les tests sur la maquette et à modifier le code.

Les grandes étapes de l'installation de PTAM sous Ubuntu sont précisées en annexe.

Sous Mac OS X

L'installation d'OpenCV s'est déroulée relativement simplement, et a été vérifiée via le l'installation et le test de KLT.

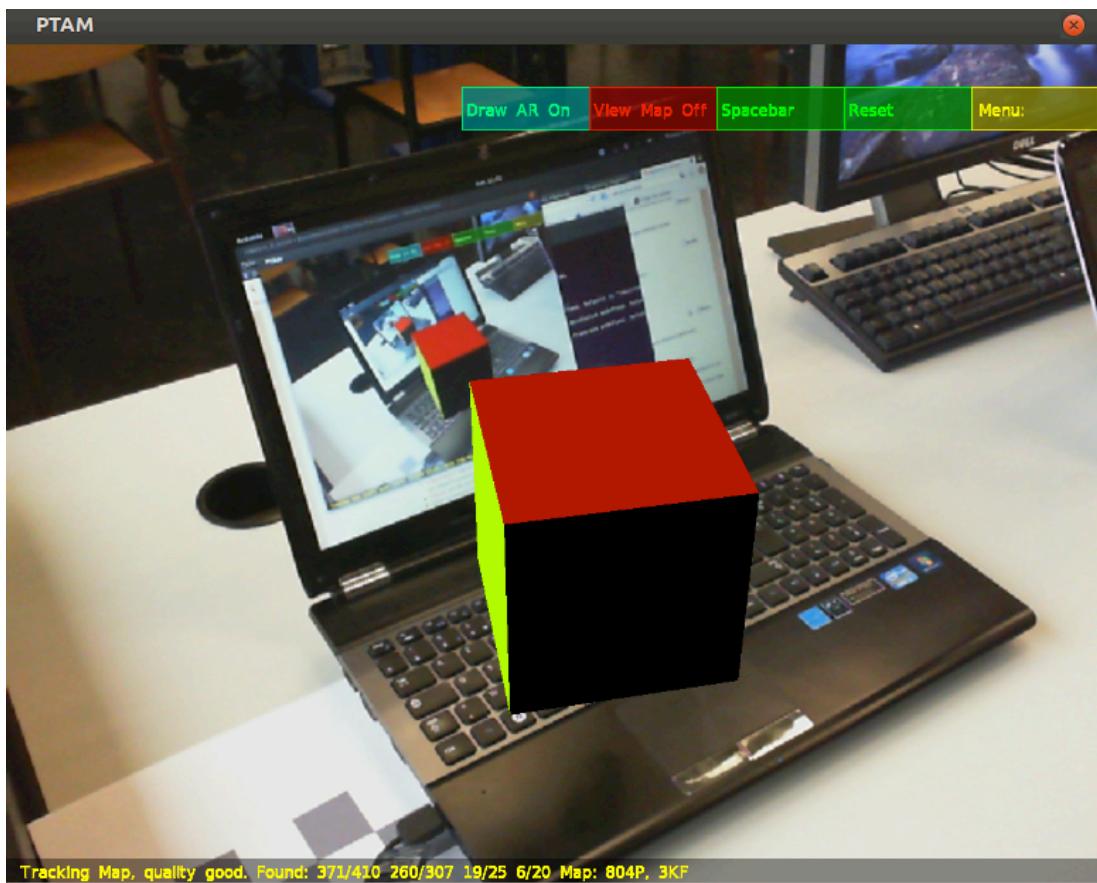
Tout comme sous Linux, afin d'installer correctement PTAM, il a fallu installer quelques bibliothèques en amont (libCVD, TooN, GVars3). En se basant sur le tutoriel fournit dans le code source de PTAM, nous avons donc téléchargé les trois bibliothèques. L'installation de TooN fut assez triviale. Cependant, l'installation des deux autres bibliothèques fut un échec sous Mac OS X 10.8. Nous avons donc essayé, à l'instar des tutoriels présents sur le web, une installation sous 10.5 qui s'est soldée par une erreur de buffer. Nous avons enfin essayé l'installation sous 10.6 et là aussi, bien que l'installation des trois bibliothèques se soit très bien passé (mais a demandé quelques heures de compilation), l'installation de PTAM a retourné une erreur de buffer. Suite à cela, nous avons donc décidé de nous focaliser sur GNU/Linux.

Découverte et prise en main des outils

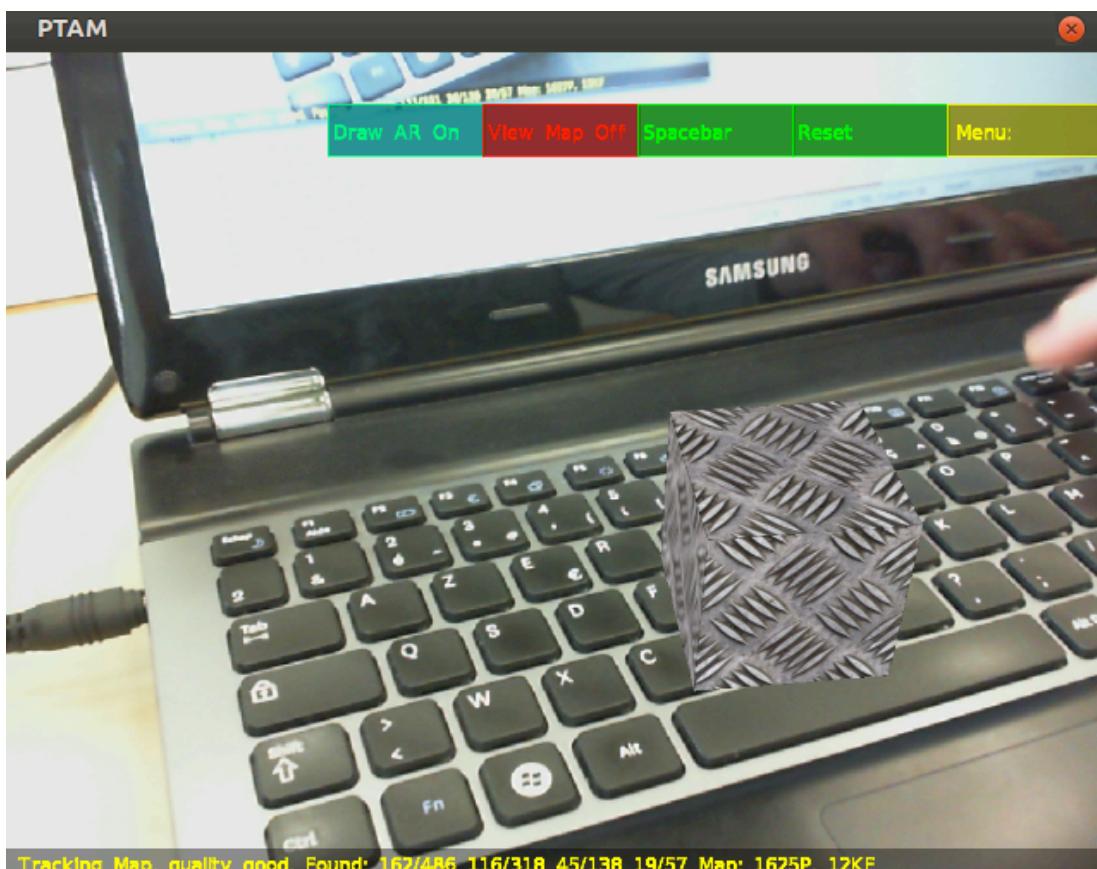
Ensuite nous avons dû nous familiariser avec ces outils, afin de les utiliser pour notre projet. Afin d'utiliser PTAM convenablement, nous avons dû apprendre les bases d'OpenGL, car PTAM est basé sur les bibliothèques d'OpenGL. La familiarisation avec OpenGL a demandé un certain temps, bien qu'elle soit codée en C++. L'utilisation d'OpenCV ne n'est donc pas avéré nécessaire puisque PTAM arrivait très bien à faire seul ce que l'on désirait.

Ajout d'informations

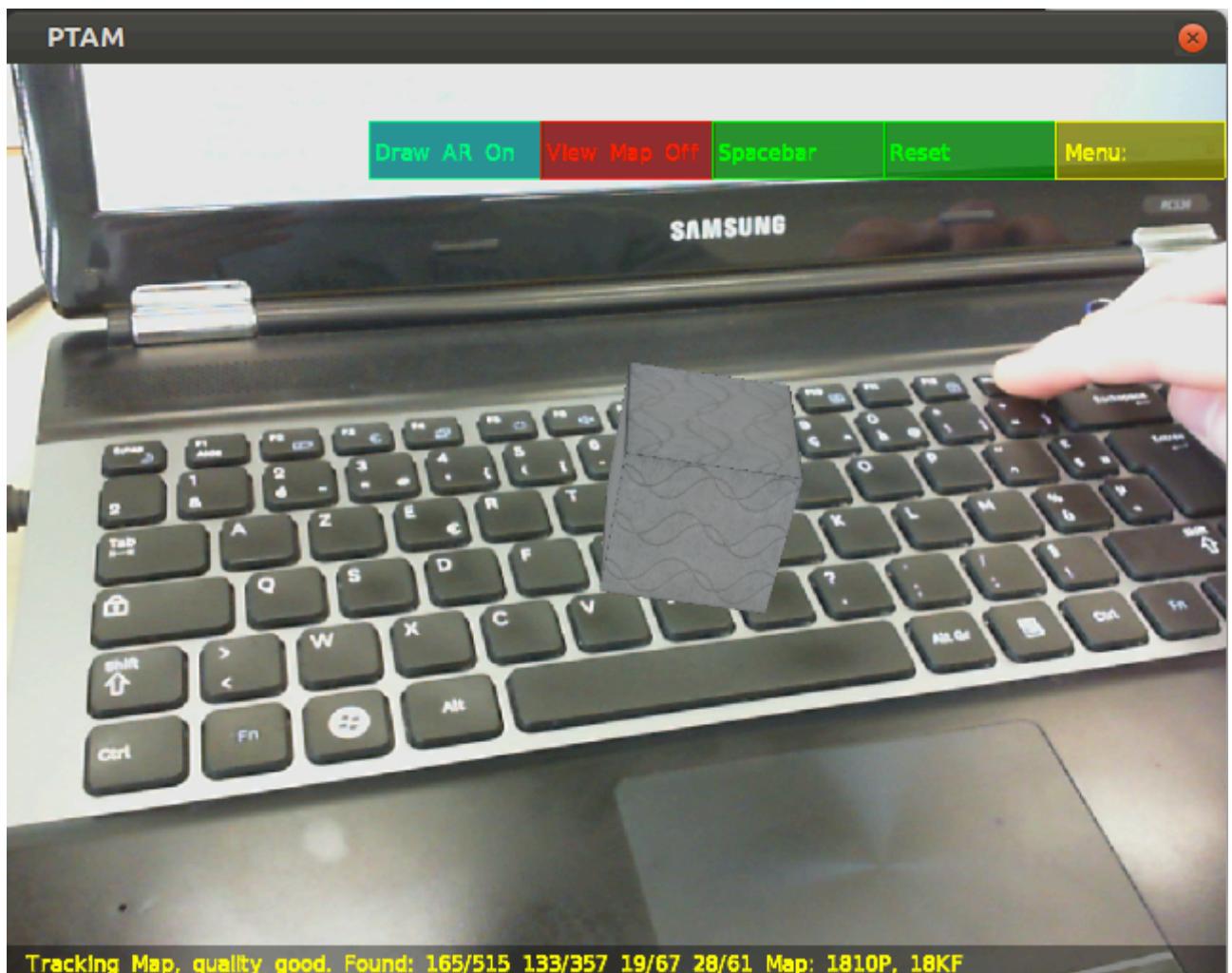
Une fois PTAM installé, nous l'avons donc testé sur la maquette. En observant les yeux affichés par le code, nous avons décidé, plutôt que de schématiser les bâtiments manquants, via Google SketchUp par exemple, de modifier le code afin que celui-ci nous affiche directement les objets. Pour cela, nous avons modélisé, grâce à OpenGL les bâtiments face par face, en se servant d'image satellites pour déterminer leurs emplacements. Une fois cela fait, il a fallu ajouter les textures via l'outil SOIL, basées sur des photos prises par nos soins. La schématisation des bâtiments et l'ajout de texture nous a demandé plusieurs heures de travail, car bien que répétitive, cette tâche ne tolérait aucune erreur sous peine d'observer des faces incomplètes voire inexistantes et dont la texture ne correspondait pas toujours à celle précisée dans le code.



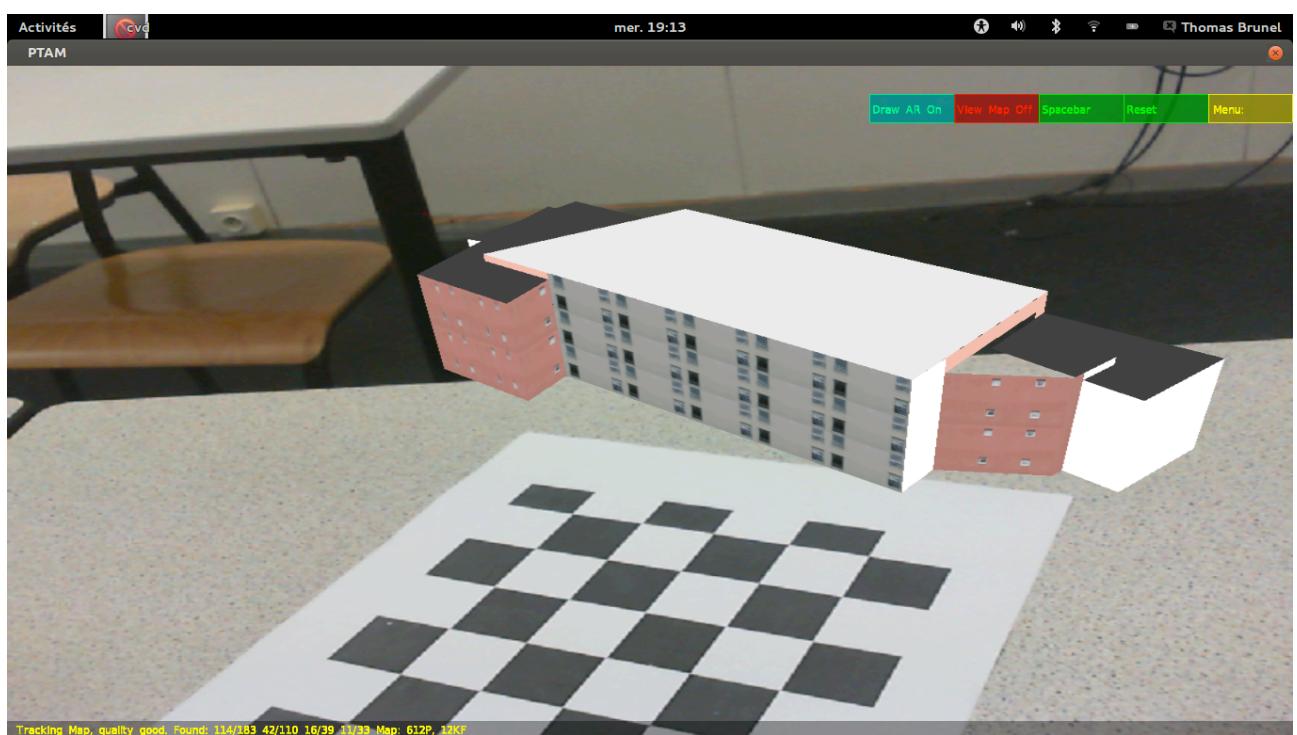
Exemple d'un cube modélisé avec PTAM



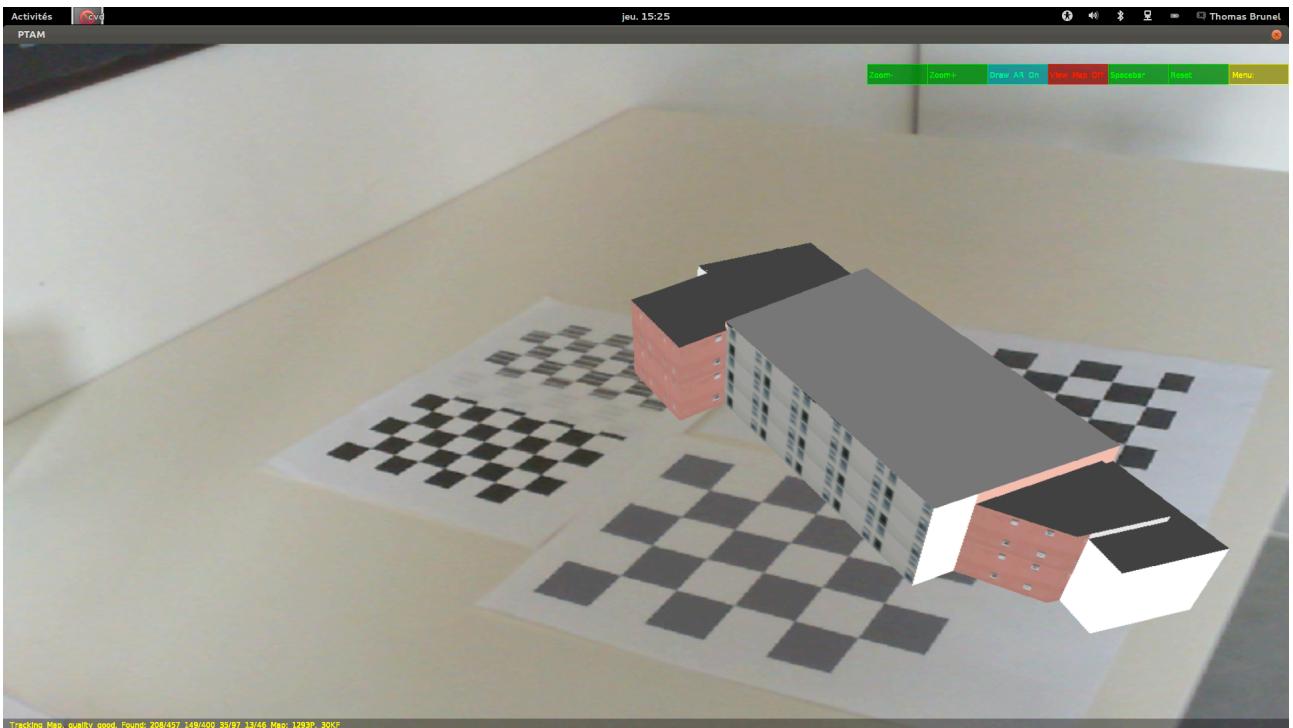
Exemple d'un cube avec une texture



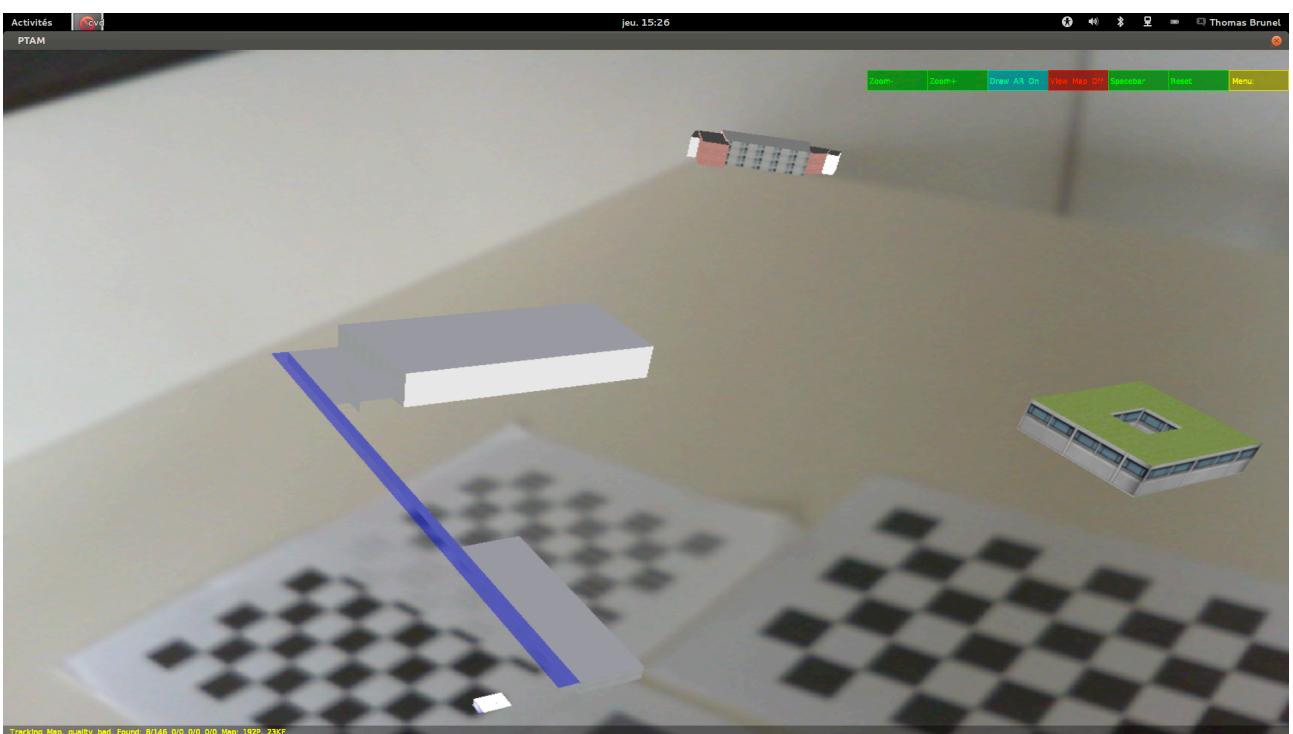
Exemple d'un cube utilisant la texture du bâtiment des carènes



Exemple de la résidence n°2 avec une erreur dans l'emplacement des faces



La résidence n°2 complètement modélisée



Les trois bâtiments modélisés avec leurs textures

Tests grandeur nature et corrections

Il est ainsi venu le temps de tester sur la maquette réelle, afin de voir si l'emplacement des bâtiments ainsi que leur échelle collaient parfaitement. Nous nous sommes heurtés à un problème quasi insolvable : la plaque de Plexiglas protégeant la maquette. En effet, le plan créé par PTAM se situe sur cette plaque (PTAM la détecte à cause des très nombreux reflets). Il a donc fallut essayer de ruser, en modifiant l'emplacement des bâtiments, mais en perdant en robustesse.

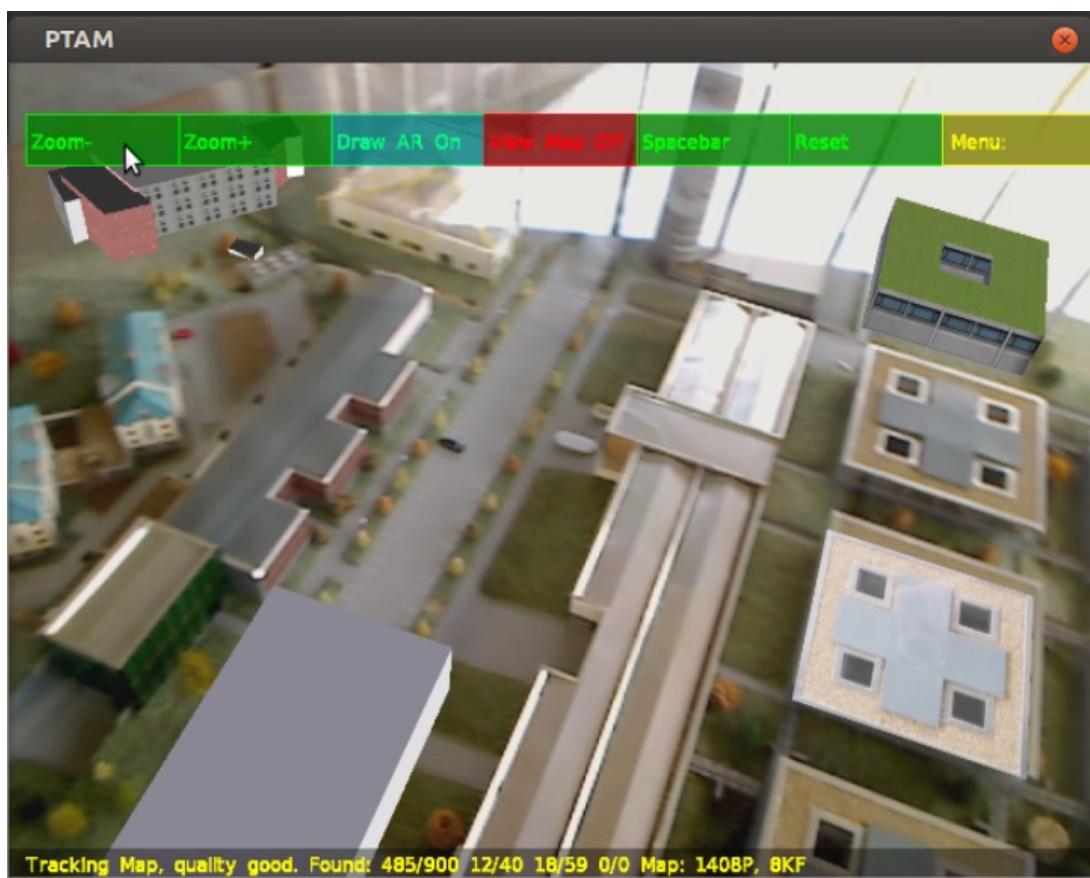
Enfin, pour faciliter l'étape de calibration, plusieurs ajouts au code ont été effectuées. D'abord, deux boutons "Zoom+" et "Zoom-" permettent de régler l'échelle des bâtiments, qui varie d'une calibration à l'autre. Ces boutons remettent à l'échelle les structures en temps réel, lorsque le programme est exécuté. De plus, l'élaboration du plan de construction se fait en partant toujours d'un bâtiment de référence : le bâtiment E. Pour systématiser cette étape, nous avons ajouté un cadre bleu fixe par rapport à l'écran, qu'il faut faire correspondre à la face du dessus du bâtiment E. Nous avons aussi modélisé cette face, pour régler avec une plus grande précision les coordonnées des objets virtuels.



Ajout des bâtiments sans réglages



Réglage des bâtiments en cours



Ajout des bâtiments une fois les réglages terminés (1)



Ajout des bâtiments une fois les réglages terminés (2)

Solution adoptée

La solution adoptée au projet permet d'afficher à l'échelle de la maquette et en temps réel le bassin des carènes, le bassin des houles, la 2eme résidence Max Schmitt, et le nouveau bâtiment en construction.

Scénario d'utilisation

Le scénario d'utilisation est le suivant :

- Calibrage de la caméra, avec un pavage en noir et blanc, avec le programme *CameraCalibrator*.
- Lancement de PTAM.
- Ouverture du flux vidéo de la webcam.
- Reconstruction du plan horizontal de la maquette, en repérant des points particuliers et en les suivant lorsque la webcam se déplace.
- En parallèle, l'analyse vidéo doit permettre de déterminer la pose de la caméra,,.
- Affichage des bâtiments modélisés auparavant, en projetant les faces visibles des objets sur le cadre d'écran, tout en gérant les occlusions (des bâtiments virtuels seulement) avec un Z-buffer.
- Remise à l'échelle des objets virtuels.
- Suivi des points lors du déplacement de l'utilisateur.

Bilan critique

Afin que le programme fonctionne correctement, il semble impératif qu'il n'y ait pas de vitrage sur la maquette. En effet, ne pouvant pas démonter la vitre de protection de la maquette de l'Ecole, PTAM créait systématiquement le plan de la réalité augmentée dans le plan de la vitre en Plexiglas. De plus, pour avoir une modélisation identique à chaque coup, il est important de calibrer la caméra strictement au même endroit, d'où notre utilisation d'un carré bleu.

Nous avons également pensé à rajouter des animations. Malgré une implémentation relativement complexe, nous avons réussi à faire des choses simples (un carré tournant sur lui-même). Cependant, l'ajout d'une animation consomme énormément de RAM rendant ainsi le programme particulièrement lent (ce phénomène empire lors de l'ajout de textures). Nous obtenons alors quelque chose qui est loin d'être du temps réel, avec un décalage de l'ordre de la minute entre ce que voit la caméra et ce que l'on voit à l'écran.

Idées d'amélioration :

La gestion des occlusions entre le réel et le virtuel

Le problème de l'occlusion en réalité augmentée a déjà trouvé des solutions, mais celles-ci sont-elles applicables à notre projet ?

On peut voir dans la vidéo suivante <http://www.youtube.com/watch?v=0XQtxmyaMR4> qu'avec ARToolKit et OpenGL, il est possible de faire de la gestion d'occlusion pour de la réalité augmentée avec marqueur. Une solution possible est de modéliser l'environnement, puis de la rendre invisible à l'affichage. On est ramené à un problème d'occlusion entre deux objets virtuels. Cette solution n'est pas applicable dans notre cas, car l'objectif est justement d'éviter d'avoir à modéliser la maquette réelle en entier.

Un travail d'occlusion a déjà été réalisé avec PTAM (<http://www.robots.ox.ac.uk/~gk/publications/KleinDrummond2004ISMAR.pdf> section 8).

Le Z-buffer fait une estimation de la profondeur du monde réel environnant les objets virtuels. Ces données peuvent être générées en temps réel. Pour cela, l'article ci-dessus fait référence à un autre travail (M. M. Wloka and B. G. Anderson. Resolving occlusion in augmented reality, *In Proc. Symposium on Interactive 3D Graphics*, pages 5-12, New York, April 1995). Ce document n'est pas accessible gratuitement.



Exemple d'occlusion de la réalité dans le monde virtuel

La gestion des occlusions avec le monde réel est une bonne idée d'amélioration, et pour cela, il faudrait avoir une bonne compréhension de l'intégralité du fonctionnement de PTAM, pour travailler sur les points détectés.

Amélioration de la stabilité

Pour plus de stabilité du modèle, on pourrait utiliser un suivi de points du bâtiment que l'on utilise pour faire le calibrage du programme (toit du bâtiment E dans notre cas). Ce suivi pourrait être réalisé à l'aide d'un autre logiciel ou une autre bibliothèque telle que OpenCV.

Pour voir plus loin

Ajout d'une ambiance sonore

Le but final de ce projet étant d'être utilisé sur une maquette historique, on peut imaginer qu'il serait intéressant, pour l'expérience utilisateur, d'ajouter une ambiance sonore en fonction de ce que l'on modélise.

Schématisation de l'évolution au cours du temps

Toujours dans l'optique de l'utilisation sur une maquette historique, on peut également imaginer l'ajout d'animation montrant l'évolution de certains bâtiments dans le temps (construction, destruction, projet d'évolution...).

Bibliographie

Le traité de la réalité virtuelle, sous la direction de Philippe Fuchs et la coordination de Guillaume Moreau.

Site de PTAM : <http://www.robots.ox.ac.uk/~gk/PTAM/>

Blog de PTAM : <http://ewokrampage.wordpress.com/>

Documentation de PTAM : http://maptopixel.com/projects/ptam_dxygen/doxygen_ptam_j/doxygen/html/annotated.html

Le site du Zéro (tutoriels) : <http://www.siteduzero.com/>

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

Annexes

Annexe 1 : Installation de PTAM sous Ubuntu 12.10

On suppose ici que les conditions matérielles requises sont obtenues (comme la carte graphique Nvidia). Par ailleurs, pour utiliser une carte « Nvidia Optimus » sous Linux, l'installation d'un autre logiciel est nécessaire ; installer par exemple Bumblebee « Nvidia Optimus support » (<https://wiki.ubuntu.com/Bumblebee>). Si vous avez fait cela, les paramètres de votre écran sont sûrement en 640*480. Pour les remettre à la bonne définition, vous pouvez aller dans les paramètres système d'Ubuntu et changer l'affichage.

Pour télécharger PTAM, aller sur : <http://www.robots.ox.ac.uk/~gk/PTAM/download.html>. Ensuite, il faut suivre le tutoriel : <http://www.robots.ox.ac.uk/~gk/PTAM/README.txt>. Comme c'est indiqué, il faut télécharger et installer les bibliothèques TooN, libCVD et Gvars3. Pour TooN, on peut la trouver sur internet : <http://www.edwardrosten.com/cvd/toon.html>. Pour libCVD aussi : <http://www.edwardrosten.com/cvd/>. De même pour Gvars3 : <http://www.edwardrosten.com/cvd/gvars3.html>. Le tutoriel indique aussi d'autres bibliothèques, et leur version -devel : blas, lapack, libgfortran, ncurses, libreadline, libdc1394 et pour une webcam firewire, libtiff, libjpeg, libpng. Normalement, elles sont toutes installées de base ou avec les trois bibliothèques principales ci-dessus. Sinon, la page <http://www.edwardrosten.com/cvd/> propose des liens vers un bon nombre d'entre-elles.

Passons ensuite à l'installation des bibliothèques :

Il faut suivre la suite du tutoriel et entrer les commandes indiquées. Quand rien n'est indiqué, il faut taper dans le dossier de la bibliothèque les commandes : ./configure, puis make et enfin make install.

En cas de problèmes, vérifier que OpenGL est installé sur l'ordinateur. Le cas échéant, on peut taper les lignes de commande suivantes (pour Ubuntu) :

```
sudo apt-get install freeglut3
```

```
sudo apt-get install freeglut3-dev
```

Enfin, l'installation de PTAM en tant que tel se fait en allant dans le dossier PTAM et en tapant la commande make. Penser à prendre le « bon » makefile, c'est à dire celui modifié lors du projet, puis de l'adapter en fonction de la configuration de l'ordinateur. L'utilisation de PTAM est assez simple, on peut se référer au tutoriel, ou à la vidéo suivante : http://www.youtube.com/watch?v=Ell_5SaNPMM.

Annexe 2 : Exemples de code

La fonction qui nous a pris le plus de temps est celle qui construit toutes les faces des objets modélisés : *DrawBatiment*, dans *batiment.cc*. On retrouve dans cette fonction des structures qui reviennent plusieurs fois : le chargement d'une structure, et la construction d'une face. Le chargement des textures se fait avec SOIL (*SOIL_load_OGL_texture*), la construction des faces avec OpenGL.

```
void Batiment::DrawBatiment() //Construction des faces des bâtiments
{
    GLuint tex_2d = SOIL_load_OGL_texture //Chargement de la texture du
bassin des carènes
    (
        "carene.png",
        SOIL_LOAD_AUTO,
        SOIL_CREATE_NEW_ID,
        SOIL_FLAG_MIPMAPS | SOIL_FLAG_INVERT_Y | SOIL_FLAG_NTSC_SAFE_RGB |
SOIL_FLAG_COMPRESS_TO_DXT
    );
    (...)

    glEnable(GL_DEPTH_TEST); //Active le Z-buffer
    (...)

    //Créations des faces :

    // Face du batiment de référence pour le calibrage
    glColor3ub(255,255,255); //La couleur derrière la texture est blanche pour ne pas interférer
    glEnable(GL_TEXTURE_2D); //Active l'utilisation de textures
    glBindTexture(GL_TEXTURE_2D, batimentRef); //On attache la texture aux éléments qui suivent
    glBegin(GL_QUADS); //Face quadrangle
    glTexCoord2d(0,1); glVertex3d(-2.5,2.5,1.5); //glTexCoord2d désigne les coordonnées de la texture à attacher au point construit ensuite.
    glVertex3d(0,0); glVertex3d(-2.5,-2.5,1.5);
    glTexCoord2d(1,0); glVertex3d(2.5,-2.5,1.5);
    glTexCoord2d(1,1); glVertex3d(2.5,2.5,1.5);
    glEnd(); //Fin de construction des quadrangles (Pour changer de texture, on est obligé de s'arrêter puis de recommencer)
```

Pour l'ajout d'un carré bleu à l'écran, avant de lancer la construction du plan, celui-ci se fait grâce à l'appel de la fonction *DrawSquare()* définie dans le fichier *GLWindow2.cc* :

```
void GLWindow2::DrawSquare()
{
    glColor3f(0,0,1); //Bleu pur
    glBegin(GL_LINE_LOOP); //Crée une ligne qui revient sur le premier
                           //point à la fin
    glVertex2i(mirVideoSize.x/2 - Square_Size,mirVideoSize.y/2 - Square_Size);
    glVertex2i(mirVideoSize.x/2 - Square_Size,mirVideoSize.y/2 + Square_Size);
    glVertex2i(mirVideoSize.x/2 + Square_Size,mirVideoSize.y/2 + Square_Size);
    glVertex2i(mirVideoSize.x/2 + Square_Size,mirVideoSize.y/2 - Square_Size);
    glEnd();
}
```

Cette fonction est appelée dans le fichier *system.cc* si le calibrage n'est pas réalisé :

```
if(!mpMap->IsGood())
    mGLWindow.DrawSquare();
```