ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Section de Microtechnique, 7ème semestre

Projet de Semestre

Réalité augmentée sur systèmes embarqués, étude des outils disponibles et des performances possibles

Grégoire Salamin
Laboratoire d'Automatique

Professeur : Denis Gillet
Assistants : Damien Perritaz et Christophe Salzmann

Date : 15.02.2007

Résumé

Ce document présente les résultats obtenus lors d'un projet de semestre à l'EPFL. Ce rapport constitue un guide de mise en route pour les personnes désirant mettre en oeuvre une application de réalité augmentée destinée aux systèmes embarqués

L'étude les différents outils disponibles pour le développement à permis de recommandé *DriectShow* pour l'acquisition vidéo, *ARToolkit Plus* pour le *tracking* et la combinaison de *Vincent* (*OpenGL ES*) et de *Glut ES* pour le rendu et la gestion de l'interface utilisateur.

Un appareil E-Ten M600 a également été utilisé pour évaluer les performances possibles. Les résultats pour les parties d'acquisition vidéo et de *tracking* n'ont pas donné satisfaction. Cependant, l'utilisation conjointe d'*OpenGL ES* et de *Glut ES* a permis un rendu performants.

Table des matières

Chapitre 1

Introduction

1.1 But du projet

Dans le cadre du projet de recherche « 6^{th} *sense* », le laboratoire d'automatique de l'EPFL a émis le souhait d'obtenir les compétences nécessaires pour la réalisation d'applications de réalité augmentée sur systèmes embarqués. Ce projet de semestre entre dans ce cadre pour évaluer les performances possibles et les solutions envisageables.

Des applications de réalité augmentée ont déjà vu le jour au sein du *Studierstube*[1] de l'université de Graz. Ce projet de semestre vise à valider les solutions trouvées par ce groupe de travail. Le cas échéant, il se propose d'expérimenter d'autres procédés dans le but d'acquérir un savoir faire dans le domaine de la réalité augmentée sur systèmes embarqués.

1.2 Qu'est-ce que la réalité augmentée ?

Un système de réalité augmentée est un système de vision permettant de superposer des objets en trois dimensions sur une image du monde réel (provenant généralement d'une caméra).

Les objets tridimensionnels sont intégrés en tenant compte du point de vue de la caméra dans l'environnement. Ils donnent ainsi l'illusion qu'il sont présents dans la scène filmée. La Figure 1.1 donne un aperçu des possibilités offertes par la réalité augmentée.

1.3 Qu'entend-t-on par systèmes embarqués ?

Un système embarqué désigne un système électronique autonome piloté par un logiciel. Dans le cadre de ce projet, les termes « systèmes em-

[1] *Studierstube* est un groupe de travail initié à l'université de Vienne en 1996 et supporté depuis 2004 par l'université de Graz. Il cherche à développer des applications mobiles de réalité augmentée.

FIG. 1.1 – ARTHUR, un projet utilisant la réalité augmentée pour venir en aide aux architectes.

barqués » désignent les organiseurs personnels (*PDA*, *PocketPC*) et les téléphones mobiles évolués (*PocketPhone*). La Figure 1.2 présente deux appareils considérés comme des systèmes embarqués.

FIG. 1.2 – A gauche : un *PocketPC* DELL Axim X51v. A droite : un *PocketPhone* HPC TyTN.

Chapitre 2

Matériel et environnement de développement

Pour créer une application de réalité augmentée sur système embarqué, il faut disposer d'un ordinateur de poche et d'une caméra. Le choix de ce matériel doit être dirigé vers un appareil disposant d'un processeur puissant, d'un écran de bonne taille et d'une caméra capable de fournir des images de bonne qualité à une fréquence adéquate. Différentes configurations ont été testées lors de ce travail et seront présentées dans ce chapitre.

2.1 HP iPaq hx4700, Windows PocketPC 2003

L'iPaq hx4700 de Hewlett Packard est l'appareil disposant du processeur le plus puissant dans l'offre actuel des assistants personnels. Il possède un processeur Intel XScale PXA270 624MHz et un écran VGA [1] 4", 65 536 couleurs (Attention : les écrans VGA chargent plus lourdement les processeurs). Pour profiter pleinement des performances du processeur, Intel fournit *Graphics Performance Primitives (GPP)*, une librairie permettant l'utilisation de fonctions graphiques 3D optimisées, voir [2].

Pour l'acquisition d'images, la caméra externe 003A de Spectec a été utilisée. Elle se branche sur le port SD de l'iPaq.

Cet assistant personnel est livré avec *Windows PocketPC 2003* comme système d'exploitation. Microsoft fournit un environnement de développement gratuit pour cette plateforme. Il s'agit de *eMbedded Visual C++ 4.2* accompagné du *SDK* [2] *for Windows Mobile 2003-based Pocket PCs*.

La limitation rencontrée avec cet environnement est l'impossibilité d'acquérir des images de la caméra. En effet, le constructeur de la caméra ne met pas

[1] Video Graphics Array : Standard d'affichage pour ordinateurs désignant une résolution de 640 x 480 pixels.

[2] SDK est l'acronyme de *Software Developpement Kit*. Il s'agit d'un ensemble de librairies et d'outils mis à disposition des développeurs.

de SDK à disposition des développeurs. La solution pour palier à ce manque est de migrer l'iPaq vers une nouvelle version du système d'exploitation, *Windows Mobile 5.0*.

FIG. 2.1 – HP iPaq hx4700 et caméra Spectec 003A.

2.2 HP iPaq hx4700, Windows Mobile 5.0

Le choix d'une mise à jour du système d'exploitation pour pallier au problème d'acquisition est motivé par les nouvelles fonctionnalités apportées par *Windows Mobile 5.0*. De nouvelles API [3] DirectX sont disponibles, dont DirectShow. DirectShow est une API déjà présente dans DirectX pour ordinateur de bureau. Elle permet d'interfacer n'importe quelle caméra dont les drivers ont été correctement installés.

Le développement pour *Windows Mobile 5.0* nécessite un nouvel environnement de développement. Il n'est plus possible d'utiliser *eMbedded Visual C++* et il faut désormé posséder *Visual Studio 2005* pour pouvoir installer *Windows Mobile 5.0 SDK for Pocket PC*.

Les drivers de la caméra Spectec pour cette nouvelle version du système d'exploitation se sont avérés inutilisables. En étudiant l'offre des caméras externes pour organiseur personnel, cet accessoire s'est avéré obsolète. Beaucoup de nouveaux appareils (*PocketPC*, *SmartPhone*) intègrent actuellement un appareil photo numérique. Il est donc préférable d'opté pour un nouvel appareil sur lequel le driver est déjà installé.

[3] API est l'acronyme d'*Application Programming Interface*. « Langage ou format de message utilisé par un programme applicatif pour communiquer avec le système d'exploitation ou un autre programme de contrôle. » ESPACES & RESEAUX NUMERIQUES, *Glossaire*, http ://www.ern.fr/index.php ?map=glossaire

2.3 E-Ten M600, Windows Mobile 5.0

L'E-Ten M600 est un *PocketPhone* ou organiseur communiquant. Le choix de cet appareil est le fruit d'une recherche visant à réunir les caractéristiques optimales pour le développement d'une application de réalité augmentée. Elles sont les suivantes :

- Processeur le plus puissant possible (comparable à l'Intel XScale PXA270 624MHz)
- Grand écran, si possible 4"
- Appareil photo numérique, résolution minimum de 320 x 240 pixels à une fréquence de 20 fps (*frame per second*)
- Accélérateur graphique (voir "Implémentation matériel", section 3.3.1)

Un comparatif de l'offre actuel a été effectué et les résultats sont présentés dans l'Annexe A.

L'E-Ten M600 offre un bon compromis des différentes exigences. Il dispose d'un processeur Samsung S3C2440 400 MHz, d'une caméra intégrée 1.3MPixel et d'un écran 2.8" d'une résolution de 320 x 240 pixels.

FIG. 2.2 – E-Ten M600.

Chapitre 3

Etude des outils disponibles pour la réalisation d'une application de réalité augmentée

Studierstube a déjà acquis beaucoup d'expérience dans le domaine de la réalité augmentée sur systèmes embarqués. Une première approche à donc consisté a évaluer leur solution en la mettant en pratique. Les librairies nécessaire étaient disponible sur leur site internet.

La solution présentée n'a pas donné satisfaction. A part *ARToolKit Plus*(voir la suite de ce chapitre), les librairies se sont avérées inadaptées. Elles n'étaient pour la plus part plus maintenues, les versions étais anciennes et parfois incompatibles avec la plateforme *Windows Mobile 5.0*. Elles étaient peu ou mal documentées.

Sur la base de cette expérience, ce chapitre traite des différentes fonctions qu'une application de réalité augmentée doit réaliser. Pour chaque fonction seront évaluées les différentes solutions envisagées dont celles issues de *Studierstube*.

3.1 Acquisition vidéo

L'acquisition vidéo doit permettre à l'application de disposer d'une image actuelle et de suffisamment bonne qualité pour permettre son traitement.

3.1.1 DirectShow

DirectShow est une API permettant de gérer des flux vidéo et audio provenant de périphériques de capture ou de fichiers multimédias. Basée sur un système de filtres, elle est totalement extensible grâce à la possibilité de

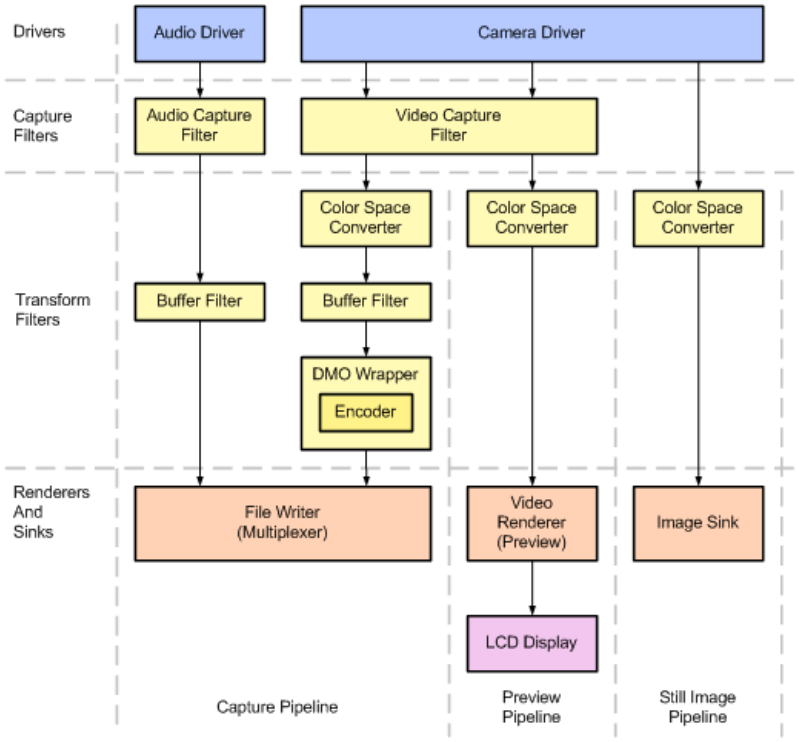

FIG. 3.1 – Architecture d'un *Filter Graph* typique pour un driver de caméra disposant de trois *PIN*.

créer ses propres composants. *DirectShow* est intégrée à *Microsoft Windows* pour ordinateur de bureau depuis de nombreuses années mais sa version pour systèmes embarqués est apparue avec *Windows Mobile 5.0* en mai 2005.

Le fonctionnement de *DirectShow* repose sur une juxtaposition de filtres élémentaires de trois types différents. Les filtres sources constituent le point d'entrée du flux multimédia qui peut provenir d'un fichier, d'une carte d'acquisition ou d'une caméra. Aux filtres sources peuvent succéder un ou plusieurs filtres de transformation. Leur rôle est de traiter le flux pour par exemple le compresser ou, lorsqu'il s'agit de vidéo, pour changer d'espace de couleur (p. ex. : YUV vers RGB). Le dernier type de filtres est celui de rendu. Ce dernier étage permet habituellement d'afficher le flux à l'écran ou d'en sauvegarder une séquence dans un fichier image ou vidéo.

La gestion des filtres se fait via le *GraphBuilder* qui construit le *Filter Graph* [1] du flux. Il garantit que l'entrée d'un filtre soit compatible avec la sortie du précédent et, le cas échéant, insère les filtres de transformation nécessaires. Un outil supplémentaire pour la capture vidéo est disponible. Il s'agit du *Capture Graph Builder*. Il facilite la création du *Filter Graph* pour

[1] Le *Filter Graph* est le schéma stockant la succession des différents filtres.

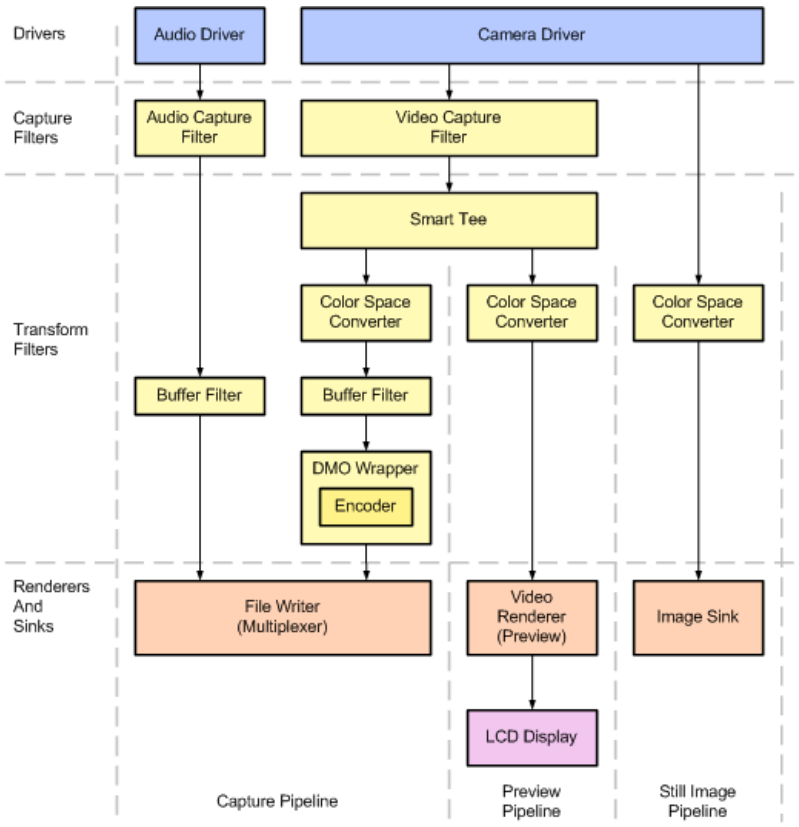

FIG. 3.2 – Architecture d'un *Filter Graph* typique pour un driver de caméra disposant de deux *PIN*.

les applications nécessitant de la capture vidéo.

L'architecture d'un *Filter Graph* de capture vidéo repose sur le driver de la caméra. Celui-ci peut exhiber deux ou trois *PIN* de sorties selon le modèle de la caméra. Un driver à trois *PIN* fournit un *PIN_CATEGORY_CAPTURE* pour l'acquisition de vidéo, un *PIN_CATEGORY_PREVIEW* pour la prévisualisation de la vidéo et un *PIN_CATEGORY_STILL* pour l'acquisition d'images (photos). Les drivers à deux *PIN* n'ont pas de *PIN_CATEGORY_PREVIEW* c'est pourquoi le *Capture Graph Builder* insère un *Smart Tee Filter* qui permet de simuler ce *PIN*. Ces deux architectures sont présentées dans les Figures 3.1 et 3.2.

Le type de média ainsi que le format d'un flux sont défini par un *Media Type*. Cette information est définie dans une structure *AM_MEDIA_TYPE* constituée d'un type principale (p. ex. *MEDIA_TYPE_Video*), d'un sous-type (p. ex. *MEDIASUBTYPE_RGB24*). Il est également possible de spécifier la taille d'image désirée ainsi que d'autres options grâce au pointeur *pbFormat*.

Pour pouvoir traiter les images provenant de la caméra, il faudra donc

constituer le bon *Filter Graph* en associant les filtres sources, de transformation et de rendu adéquats.

3.2 Tracking

La réalité augmentée consiste à superposer un objet tridimensionnel à une image provenant d'un capteur optique. L'objet ajouté devra posséder les proportions et l'orientation adéquates pour donner l'illusion qu'il est réellement présent dans l'environnement filmé. Il est donc nécessaire de pouvoir repérer la position de la caméra dans l'environnement, c'est le rôle du *tracking*. Le *tracking* fournit une matrice permettant de projeter les objets virtuels dans le repère associé à l'environnement réel.

Plusieurs techniques existent pour obtenir une telle matrice, la détection de volume, la détection de marqueur et la vision stéréoscopique. La détection de volume nécessite de reconnaître un volume présent dans l'environnement et en fonction de sa déformation d'en tirer la matrice de transformation. Cette approche est donc fortement liée à l'environnement et offre donc des perspectives d'application limitée. La vision stéréoscopique traite les images provenant de deux caméras filmant la même scène pour en extraire l'information désirée. Cela n'est pas envisageable pour ce travail en raison des limitations matérielles imposées par l'utilisation de *PocketPC*.

C'est donc la détection de marqueur qui a été choisie pour ce travail. Cette approche consiste à détecter dans l'image filmée la présence d'un ou de plusieurs marqueurs. Dans notre cas, un marqueur se compose d'un cadre carré de couleur noir entourant une image binaire représentant un identifiant unique. Différents marqueurs sont représentés sur la Figure 3.3.

FIG. 3.3 – Exemples de marqueurs. En évidence sur le premier, l'identifiant unique, et sur le deuxième, le cadre.

Une fois le marqueur repéré, il est identifié et la matrice de transformation est calculée sur la base de la déformation du cadre. La Figure 3.4 montre un exemple de réalité augmentée par détection de marqueur.

3.2.1 ARToolkit

ARToolKit est une librairie de réalité augmentée basée sur la détection de marqueur. L'algorithme de détection effectue plusieurs étapes avant d'arriver au résultat désiré. La première étape consiste en une binarisation effectuée

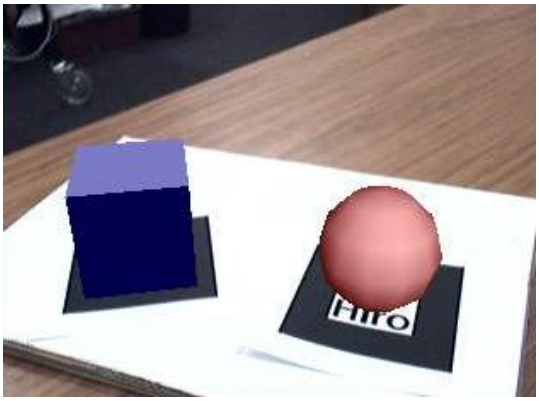

FIG. 3.4 – Exemple de réalité augmentée par detection de marqueur.

par seuillage de la luminosité de l'image fournie par la caméra. Ensuite, l'algorithme effectue une détection de contour en utilisant un *Canny Edge Detector*. Il identifie le marqueur détecté puis calcul la matrice de transformation grâce à la déformation du cadre. Les différentes étapes sont visibles sur la Figure 3.5

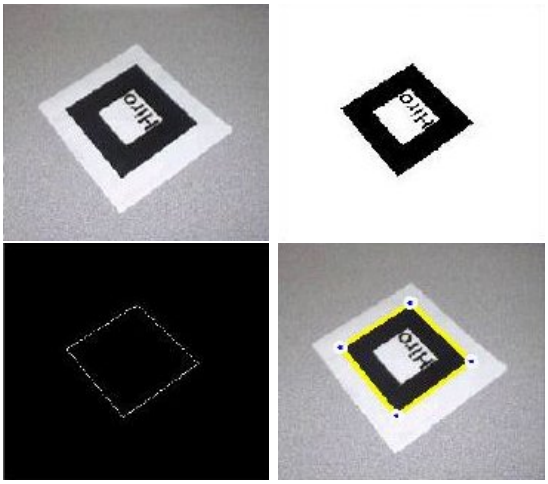

FIG. 3.5 – ARToolkit : Les différentes étapes de l'algorithme de détection.

Cette librairie n'est cependant pas adaptée à une utilisation sur systèmes embarqués, mais une version optimisée est disponible. Il s'agit d'*ARToolKit Plus*.

3.2.2 ARToolkit Plus (Studierstube)

ARToolkit Plus est un projet crée par *Studierstube*. Plus qu'une simple adaptation d'*ARToolkit* pour systèmes embarqué, cette librairie doit égale-

ment apporter de nouvelles fonctionnalités.

Pour augmenter sa rapidité de calcul, elle prend en charge l'utilisation des *Graphics Performance Primitives (GPP)* d'Intel sur les appareils dotés de processeurs XScale. Elle dispose depuis peu d'une implémentation tirant partie de l'utilisation de variables en virgule fixe semblant donner des résultats satisfaisants même sur des appareils ne permettant pas l'utilisation de *GPP*. Cette optimisation doit permettre de traiter jusqu'à 200 images par seconde (5.0 ms par traitement) sur un organiseur de poche Axim X50v de DELL. Cet appareil possède un processeur comparable à celui de l'iPaq hx4700 et est donc un peu plus puissant que celui de l'E-Ten M600.

ARToolkit Plus offre deux classes différentes au développeur : *TrackerSingleMarker* et *TrackerMultiMarker*. La première ne permet la détection que d'un seul marqueur et donc seul le marqueur reconnu avec le plus de fiabilité est retourné. La seconde renvoie les informations liées à chacun des marqueurs trouvés.

Comme mentionné sur le site de *Studierstube*, cette librairie n'est pas une solution « *out-of-the box* ». En conséquence, aucun fichier binaire n'est fournit, ni aucune information sur la configuration de l'environnement de développement. Cependant la compilation est plutôt aisée mais les variables préprocesseurs à définir pour profiter des optimisations ne sont pas claires.

3.3 Rendu

L'étape de rendu doit permettre à l'utilisateur de voir la scène de réalité augmentée. Pour cela, la librairie doit être capable d'afficher l'image provenant de la caméra comme fond et de lui superposer des objets en trois dimensions. La librairie doit être capable de projeter les objets 3D selon les informations fournies par la librairie de réalité augmentée.

FIG. 3.6 – Exemple d'application utilisant *Vincent*, une implémentation logicielle d'*OpenGL ES*.

3.3.1 OpenGL ES

OpenGL ES (Open Graphics Library for Embedded System)[3] est une spécification du *Khronos Group* définissant une API pour la génération d'images 3D sur plateforme mobile. Elle dérive de la spécification *OpenGL* et constitue une version allégée et optimisée pour les systèmes embarqués (voir [4]).

Différentes implémentations de cette API existes. Elles sont soit logicielle comme *Hybrid Rasteroid* et *Vincent*, soit *hardware* sur certains organiseurs de nouvelle génération.

Hybrid Rasteroid

Hybrid Rasteroid est une implémentation logicielle d'*OpenGL ES*. Développée par la firme *Hybrid Graphics*, elle est fournie sans ses sources et est déjà compilée. Aucune librairie n'est fournie pour la plateforme *Windows Mobile 5.0* mais la version *Windows PocketPC 2003* semble être compatible selon un message posté sur le forum de support (voir [6]). Son utilisation est limitée à certains appareils (*Acer n10/30, ASUS My Pal A620/716, Dell Axim X-series, hp iPaq h-series, ViewSonic PC V36, Toshiba e-series*). Cette librairie n'a donc pas été retenue.

Vincent

Comme *Hybrid Rasteroid*, *Vincent* est une implémentation logicielle - d'*OpenGL ES* . Livrée avec ses sources, elle peut être compilée pour *Windows Mobile 5.0* et est indépendante de l'appareil utilisé. Sa conformité avec la spécifications a été certifiée.

Implémentation matériel

L'implémentation matériel n'est disponible que sur peu d'appareil dont les *DELL Axim 50v* et *X51v*. Ces organiseurs possèdent une puce d'accélération *Intel 2700G*[7] supportant *OpenGL ES*. Ils ne disposent cependant pas de caméra intégrée et ne constituent donc pas des solutions intéressantes pour notre application.

La disponibilité d'une gestion *hardware* d'*OpenGL ES* doit cependant être prise en compte dans le choix d'un appareil destinée à la réalité augmentée.

3.3.2 Klimt (Studierstube)

Klimt est la librairie développée par *Studierstube* pour effectuer le rendu de leurs applications de réalité augmentée sur systèmes portables. Bien que fortement inspirée d'*OpenGL ES*, la conformité avec la spécification du *Khronos Group* n'a pas été respectée comme le montre le Tableau 3.1.

	OpenGL	OpenGL|ES	Klimt
glBegin/glEnd	✓	✗	✓
Prim Types	all	no Quads, Quad Strips & Polygons	all
Vertex Arrays	✓	✓	✓
Vertex Lighting	✓	✓	✓
Data Types	float, double, int, etc...	float, fixed	float, double, int, fixed
Multisampling & AA	optional	optional	✗
Polygon Stipple & Smooth	✓	✗	✗
glDraw/Read Pixels	✓	only glReadPixels	only glDrawPixels
Textures	all Types	2D	2D
Texture wrap, repeat, etc..	✓	wrap, clamp_to_edge	wrap
compressed Textures	✓	✓	✗
Multitexture	✓	optional	✗
Fog	✓	✓	✓
Scissor Test	✓	✓	✓
Alpha Test	✓	✓	✗
Stencil Test	✓	optional	✗
Depth Test	✓	✓	✓
Blending	✓	✓	✓
Clipplanes	✓	✗	✗
WGL Functions	✓	✗	✓
EGL Functions	✗	✓	✓

TAB. 3.1 – Comparaison entre OpenGL, OpenGL ES et Klimt

La compilation doit être effectuée à partir des sources fournies. Lors de la compilation pour *Windows Mobile 5.0* avec *Visual Studio 2005* des problèmes ont été rencontrés (symboles externes non reconnus, parties de code obsolète). La version la plus récente date du 16 mars 2004. Dans sa version actuelle, cette librairie n'est donc pas à recommander.

3.3.3 Direct3D Mobile

Direct3D existe déjà sur ordinateur de bureau et fait partie de *Microsoft DirectX*. Sur cette plateforme, *Direct3D* est en concurence avec *OpenGL*. Sur systèmes embarqués, Direct3D a fait son apparition avec *Windows Mobile 5.0*.

Il s'agit donc d'une API récente et peu expérimentée sur systèmes embarqués. De plus, des fonctions d'*ARToolkit Plus* sont prévue pour faciliter la cohabitation avec une librairie de rendu de la famille *OpenGL*. Pour ces deux raisons, *Direct3D Mobile* n'a pas été retenu pour ce projet.

3.4 Gestionnaire d'application

Le gestionnaire application doit permettre de traiter les interactions de l'utilisateur avec l'application (utilisation du stylet, redimensionnement). Il

doit également permettre l'intégration du rendu 3D en gérant le *double buffering*. En effet, pour éviter des effets de scintillement à l'écran, il est nécessaire d'utiliser un *backbuffer*[2] et un *frontbuffer*. Le *frontbuffer* contient l'image affichée actuellement à l'écran. Le *backbuffer* est utilisé pour construire la prochaine image qui devra être affichée. Une fois que l'image est prête, le gestionnaire échange les deux *buffers* et la prochaine image peut être dessinée dans le nouveau *backbuffer*.

3.4.1 PocketKnife (Studierstube)

PocketKnife est le gestionnaire d'application conseillé par *Studierstube*. Il a été développé par Daniel Wagner [3] à l'université de Vienne. Il supporte le *double buffering* et la gestion du stylet. Les sources sont fournies et après quelques corrections mineures la compilation s'effectue correctement dans *Visual Studio 2005*. Son utilisation avec *Klimt* a pu être testée avec succès sur *PocketPC 2003*. Cependant cette librairie n'est plus maintenue et seul une ancienne version est disponible sur le site de *Klimt*. Ce gestionnaire n'a donc pas été retenu.

3.4.2 Glut ES

Glut ES est l'équivalent pour systèmes embarqués de *Glut* (*Open**GL** **U**tility **T**oolkit*) qui est utilisé sur les ordinateurs de bureau. Comme son nom l'indique, cette librairie a été créée pour fonctionner avec OpenGL ES dont elle facilite l'utilisation. Elle permet grâce à un système de *callbacks* [4] de gérer le stylet, le redimensionnement, les menus contextuels et l'affichage en *double buffering*. Les sources sont fournies et doivent être compilées. La Figure 3.7 illustre les capacités de *Glut ES*.

[2] Un buffer est un espace alloué dans la mémoire.

[3] Daniel Wagner est au coeur du projet de réalité augmentée de *Studierstube*. Il a participé à l'écriture de nombreuses librairies utilisées par ce groupe de travail.

[4] « En informatique, une fonction de rappel (callback en anglais) est une fonction qui est passée en argument à une autre fonction. Cette-dernière peut alors faire usage de cette fonction de rappel comme de n'importe quelle autre fonction. » WIKIPEDIA, *Fonction de rappel*, http ://fr.wikipedia.org/wiki/Callback

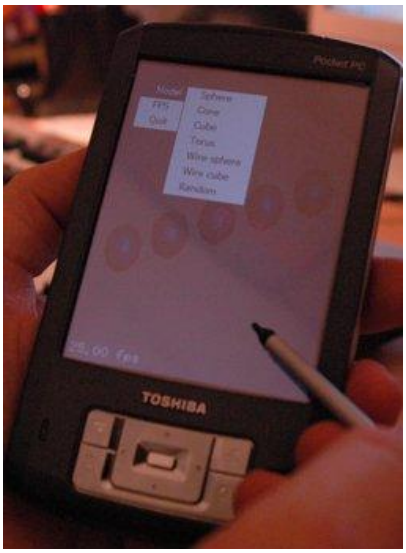

FIG. 3.7 – Exemple d'application utilisant *Glut ES* et *OpenGL ES*.

Chapitre 4

Solution retenue et résultats

Ce chapitre décrit l'application développée pour tester les performances possibles de réalité augmentée sur un appareil E-Ten M600. La solution sera jugée satisfaisante si elle permet d'obtenir un rendu d'au moins 10 images par seconde. Cela correspond à une durée d'exécution inférieure à 100 ms pour un traitement. Un traitement doit comprendre l'acquisition vidéo, la détection de marqueur et le rendu.

4.1 Structure de la solution adoptée

L'étude des solutions possibles présentées dans le chapitre précédent à conduit à la réalisation d'une application basée sur *DirectShow*, *ARToolkit Plus*, *Vincent* (qui implémente *OpenGL ES*) et *Glut ES*.

Au lancement de l'application (dans la fonction *main()*), les différentes librairies sont initialisées. L'acquisition est alors démarrée dans un *thread* secondaire et le *main()* confie la gestion de l'application à *Glut ES* dans le *thread* principale, fonction glutMainLoop().

Lorsque *DirectShow* capture une nouvelle image, il appelle la fonction de *callback* en lui passant la nouvelle image en paramètre. Cette fonction transmet l'image à *ARToolKit Plus* qui calcule la matrice de transformation. La matrice et l'image sont alors mis à disposition de Glut ES.

Périodiquement, Glut ES déclenche un appel de la fonction *display()*. Cette fonction exploite les résultats d'*ARToolKit* ainsi que l'image capturée pour effectuer le rendu grâce à *Vincent*.

Le Figure 4.1 présente le schéma de fonctionnement de l'application.

Dans les sections suivantes, la mise en oeuvre de chacune des librairies est décrite et les résultats obtenus sont présentés

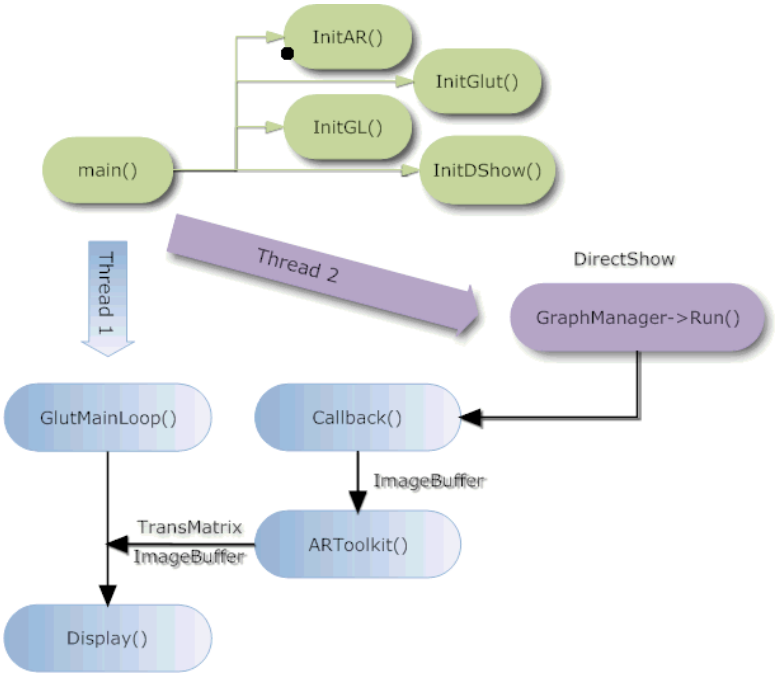

FIG. 4.1 – Schéma de fonctionnement de l'application.

4.2 DirectShow

4.2.1 Solution adoptée

Pour utiliser *DirectShow*, une classe *CGraphManager* a été créée. Elle s'occupe de construire un *Filter Graph* adapté à la capture d'images. Le filtre source est un *VideoCapture Filter*. Il est suivi d'un filtre de transformation servant à acquérir les images et d'un filtre de rendu bouclant le flux.

Filtre de transformation

Dans la version *desktop* de *DirectShow*, il existe un filtre appeler *Sample Grabber Filter*. Ce dernier est un filtre de transformation qui permet de stocker dans un buffer les images le traversant. Ce filtre répond tout à fait au besoin de l'application mais n'est malheureusement pas disponible dans la version *Mobile* de *DirectShow*. Il a donc dû être recodé pour pouvoir être utilisé.

Le codage de filtre *DirectShow* pour *Windows Mobile 5.0* n'est pas une tâche aisée. La documentation et les exemples à ce sujet sont peu abondants. Il faut donc se référer à la documentation *MSDN* [1] pour la plateforme *Win32*

[1] MSDN désigne le Microsoft Developer Network, qui « est la section de l'entreprise Microsoft qui s'occupe de ses relations avec les développeurs. » WIKIPEDIA, *MSDN*, http ://fr.wikipedia.org/wiki/MSDN

(version pour ordinateur de bureau), voir [9]. Elle offre un tutoriel intéressant intitulé « *How to get data from a Microsoft DirectShow Filter Graph* ». Il se base sur l'exemple de filtre *GrabberSample* du *Microsoft DirectX 8.1 SDK*.

Ce tutoriel a permis la réalisation d'un filtre appelant une fonction *callback* lorsqu'une nouvelle image est disponible. Le nouvel échantillon capturé est passé en paramètre à cette fonction.

Filtre de rendu

Le filtre de rendu nécessaire pour cette application est un *Null Renderer Filter*. Sont rôle est de détruire les échantillons qui lui sont transmis sans en faire le moindre traitement. Sa présence est toute fois indispensable car tout *filter graph* doit être terminé par un filtre de rendu.

Comme pour le *SampleGrabber Filter*, ce filtre n'est disponible que dans la version *Win32* de *DirectShow*.

Suite au manque de temps pour porter ce filtre sous *Windows Mobile 5.0*, le filtre de rendu par défaut à été utilisé. Il s'agit du *VideoRenderrer Filter* qui affiche le flux vidéo à l'écran.

4.2.2 Résultat

Pour évaluer les performances de ce système d'acquisition, le temps entre deux appels de la fonction *callback* a été mesuré. L'intervalle est de 300 ms ce qui correspond à une fréquence de 3.3 Hz. Ce résultat est très insuffisant. De plus, l'image retournée n'a pas la taille, ni le format souhaité. La couleur est codée en YUV [2] et la taille de l'image est de 172 x 144 pixels.

Ces résultats témoignent de la nécessité de maîtriser *DirectShow* pour pouvoir sélectionner correctement le format désiré, d'améliorer les temps d'acquisition et de développer un *Null Renderer Filter*.

4.3 ARToolkit Plus

4.3.1 Solution adoptée

Compilation

La compilation de cette librairie peut être faites avec la variable préprocesseur *_USEFIXED_* déclarée. Dans ce cas, la librairie utilisera des variables en virgules fixes censées améliorer les performances. C'est ce qui a été fait pour cette application.

[2] « Le modèle YUV définit un espace colorimétrique en trois composantes. Le premier représente la luminance et les deux autres représentent la chrominance. » WIKIPEDIA, *YUV*, http ://fr.wikipedia.org/wiki/YUV

Fonctionnement

Comme expliqué au chapitre précédent, *ARToolkit Plus* fournit deux classes différentes l'une pour la détection d'un marqueur unique et l'autre pour plusieurs marqueurs. Dans le cas de cette application, la première possibilité a été choisie. Un objet *TrackerSingleMarkerImpl* est donc instancié lors de l'initialisation de l'application.

Différents paramètres doivent également être fixés. Il s'agit de l'espace des couleurs et de la taille de l'image, du format du marqueur et du type de caméra utilisée.

Les images fournies par *DirectShow* n'ayant pas le format désiré, c'est l'image représentée sur la Figure 4.2 qui a été utilisée. Elle mesure 320 x 240 pixels.

C'est la fonction *calc()* de la classe *TrackerSingleMarkerImpl* qui calcul la matrice de transformation.

FIG. 4.2 – Image utilisée pour tester les performances d'*ARToolkit Plus*.

4.3.2 Résultats

Pour évaluer les performances de la librairie, le temps d'exécution de la fonction *calc()* a été mesuré. La durée obtenue est 180 ms ce qui correspond à 5.6 traitements par seconde.

Le résultat obtenu est très loin des 200 traitements par seconde annoncés sur le site de *Studierstube*. Le peu d'information disponible sur les conditions de cette expérience ne permet pas d'identifier clairement les raisons d'un tel écart. Il est possible cependant que l'utilisation des virgules fixes nécessite une configuration supplémentaire lors de la compilation ou de l'utilisation. Malheureusement, la librairie n'est pas documentée et mériterait donc que des recherches supplémentaires soit effectuées.

4.4 Vincent (OpenGL ES)

4.4.1 Solution adoptée

Pour ne pas influencer les performance de rendu avec les mauvais résultats de *DirectShow* et d'*ARToolkit Plus*, la détection de marqueur est effectuée lors de l'initialisation. C'est à nouveau l'image de la Figure 4.2 qui est utilisée. Ainsi *Vincent* peut utiliser les résultats calculés par *ARToolkit Plus*.

Vincent effectue le rendu 3D au sein de la fonction *display* appelée par *Glut ES*.

A chaque actualisation de l'affichage, il doit modifier ses matrices de projection (*GL_PROJECTION*) et de point de vue (*GL_MODELVIEW*). La première definie le repère objet dans lequel les objets tridimensionnels seront créés. Il faut voir une analogie entre le repère cartésien associé au marqueur (voir Figure 4.3) et le repère définit par la matrice de projection.

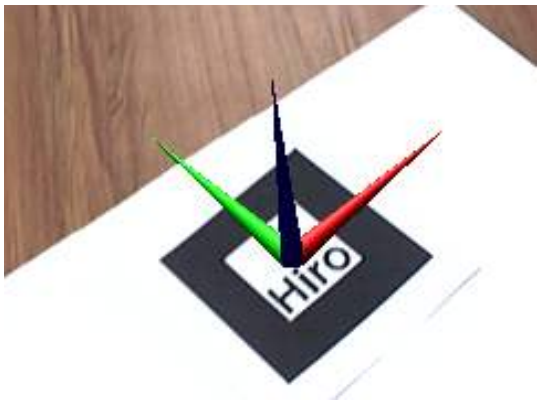

Fig. 4.3 – Repère associé à un marqueur.

La matrice de point de vue permet de positionner la caméra virtuelle filmant la scène. La position de la caméra virtuelle exprimée dans le repère objet (matrice de projection) doit être la même que celle de la caméra réelle exprimée dans le repère associé au marqueur. La Figure 4.4 présente une même scène vue dans le monde réel et dans le monde virtuel.

ARToolkit Plus fournit deux fonctions (*getProjectionMatrix()* et *getModelViewMatrix()*) permettant d'initialiser les matrices de projection et de point de vue en fonction du traitement effectué sur l'image provenant de la caméra.

Après l'initialisation des matrices, *Vincent* dessine les objets tridimensionnels. Pour cette application, il s'agit d'un cube devant être centré sur le marqueur.

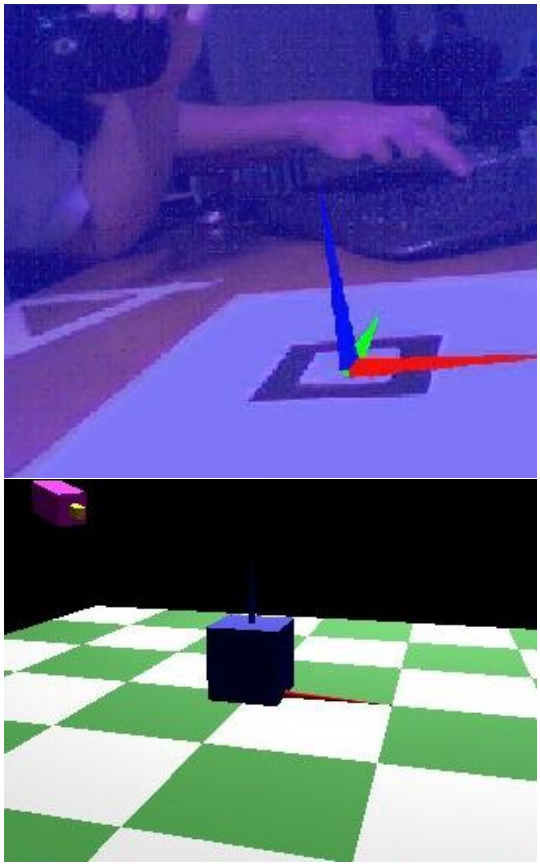

FIG. 4.4 – Illustration de l'analogie entre le monde réel et le monde virtuel 3D.

4.4.2 Résultats

Pour évaluer les performances du rendu 3D, le temps nécessaire pour initialiser les matrices et projeter le cube a été mesuré. La valeur obtenue est 20 ms. Ce résultat pris seul n'est pas très intéressant. Il est préférable de l'évaluer conjointement à *Glut ES*. Cela permettra d'obtenir une information sur le nombre d'images par seconde pouvant être affichée par l'application.

4.5 Glut ES

4.5.1 Solution adoptée

Dans cette application, *Glut ES* a été utilisé avec un minimum d'options. A part la gestion de l'affichage, seule la détection d'un appui du stylet est gérée pour pouvoir arrêter le programme.

Ces différentes fonctionnalités doivent être initialisées en associant une

fonction *callback* à chacun des événement à gérer. Par exemple, l'appel de « *glutDisplayFunc(display)* » demande à *Glut ES* d'appeler la fonction *display* à chaque fois que l'affichage doit être mis à jour.

4.5.2 Résultats

Pour évaluer les performances de *Glut ES*, la durée entre deux appels de la fonction *display* à été mesurée. De plus, l'initialisation des matrices de projection et de point de vue et l'affichage du cube ont été désactivés (voir section 4.4, Résultats). Le résultat obtenu est 20 ms.

En additionnant cette valeur avec le temps nécessaire pour l'affichage d'un cube avec *Vincent*, on obtient 40 ms. Ce résultat signifie que l'on peut espérer un rendu d'environ 25 images par seconde. Cela est tout à fait acceptable pour une application de réalité augmentée.

4.6 Synthèses des résultats

Dans l'ensemble, la solution présentée dans ce chapitre ne permet pas de réaliser une application de réalité augmentée sur l'appareil E-Ten M600.

L'acquisition vidéo avec *DirectShow* n'est pas encore maîtrisée. Le format et les dimensions de l'image capturée ne peuvent pas être choisis. L'acquisition se déroule beaucoup trop lentement.

ARToolkit fonctionne correctement mais le temps de calcul est trop important. Des essais supplémentaires sont nécessaires pour s'assurer de bénéficier de toute les optimisations fournies par cette librairie.

La combinaison de *Vincent* et de *Glut ES* offre un rendu satisfaisant. Il permet d'obtenir un résultat d'environ 25 images par seconde pour l'affichage d'un cube.

	Durée [ms]	Fréquence [Hz]
Acquisition - *DirectShow*	300	3.3
Tracking - *ARToolkit Plus*	180	5.6
Rendu - *Vincent* et *Glut ES*	40	25

Tab. 4.1 – Récapitulatif des résultats obtenus

Chapitre 5

Conclusion

5.1 Studierstube

La solution proposée par *Studierstube* est une base intéressante pour entrer dans le domaine de la réalité augmentée sur systèmes embarqués. Cependant, elle ne fournit pas les informations nécessaires pour pouvoir reproduire rapidement et simplement les résultats annoncés.

Les librairies recommandées sont mal ou peu documentées. Elles ne sont visiblement pas maintenues à jour ce qui rend leur utilisation difficile dans les environnements de développement actuels.

Seul *ARToolkit* échappe à cette règle et présente un intérêt certain. Il faudrait cependant pouvoir se pencher plus en détail sur son implémentation pour saisir le fonctionnement des optimisations disponibles. Cela permettrait peut-être de compiler une version plus performante de cette librairie et, ainsi, d'approcher les résultats annoncés.

5.2 Outils et performances

Une application de réalité augmentée peut-être découpée en quatre fonctionnalités : l'acquisition vidéo, le *tracking*, le rendu 3D et l'interface utilisateur. Le choix des outils remplissant ces fonctionnalités doit être dicté par l'optimisation du temps d'exécution qui est le facteur limitant sur les systèmes embarqués. Différents outils ont été retenus.

DirectShow

Pour l'acquisition, *DirectShow* permet d'obtenir des images provenant de la caméra. Il s'agit d'une API destinée à de nombreuses applications. Pour cette raison, sa prise en main est difficile. De plus, elle existe depuis peu sur les systèmes d'exploitation pour systèmes embarqués. De nombreuses fonctionnalités sont manquantes et doivent donc être portées depuis la version

pour ordinateur de bureau.

Les efforts fournis pour l'utiliser n'ont pas permis d'obtenir des résultats convainquants. Le temps d'acquisition était trop long et il n'a pas été possible de choisir le format de l'image capturée

Malgré cela, *DirectShow* constitue actuellement la seule solution acceptable pour exploiter les données provenant des caméras. Un effort important doit être fait pour appréhender cette API afin d'en faire un outils adapté.

ARToolkit Plus

ARToolkit Plus fonctionne avec des temps d'exécution encore trop importants. Cependant, une librairie de *tracking* est indispensable pour la réalisation d'une application de réalité augmentée. *ARToolkit Plus* est actuellement la seule existante et c'est pour cette raison qu'il faudrait s'y pencher avec plus d'attention.

Vincent et *Glut ES*

La conjonction de ces deux librairies permet d'obtenir un outils performant pour la gestion du rendu 3D et de l'interface utilisateur. Ce duo est donc à recommander pour le développement d'application de réalité augmentée sur systèmes embarqués. De plus, le fait que *Vincent* respecte la spécification *OpenGL ES* permet d'envisager de le remplacer par une implémentation *hardware* sur certains appareils.

5.3 Conclusion

Les développements réalisés lors de ce projet ne constituent pas encore une application de réalité augmentée. Cependant, ils donnent un cadre pour la recherche d'une solution acceptable.

Si la solution de rendu est satisfaisante, il n'en est pas de même pour l'acquisition vidéo et le *tracking*. Dans ces domaines primordiaux, des efforts importants doivent être fait. Ce n'est qu'a ce prix qu'une solution de réalité augmentée sur systèmes embarqués pourra être envisagée.

Lausanne, le 15 février 2007

Annexe A

Comparatif de l'offre des PocketPC/Phone au 08.01.2007

Dans le tableau ci-dessous sont présentés les différents modèles de *PocketPC* et de *PocketPhone* sur le marché au 08.01.2007. Certains critères sont éliminatifs. Ils sont présentés en rouge dans le tableau. Le choix entre les modèles restant s'est effectué en fonction de tests de performance trouvés sur internet ainsi qu'en fonction des disponibilités.

	OS	Processeur	RAM	ROM	Camera	CamReso	Ecran	Resolution	Chip 3D	Wifi
Asus										
MyPal A730	Windows Pocket PC 2003	PXA270 520MHz	64	64	1.3Mpixels			640x480		
MyPal A730W	Windows Pocket PC 2003	PXA270 520MHz	128	64	1.3Mpixels			640x480		802.11b
MyPal A636	Windows Mobile 5.0	PXA270 416MHz	64	128	N/A			240x320		802.11b
MyPal A636N	Windows Mobile 5.0	PXA270 416MHz	64	128	N/A			240x320		802.11b/g
MyPal A620	Windows Pocket PC 2003	PXA255 400MHz								
MyPal A716	Windows Pocket PC 2003	PXA255 400MHz								
MyPal A639	Windows Mobile 5.0	PXA270 416MHz		1GB	N/A			240x320		802.11b/g
MyPal A632N	Windows Mobile 5.0	PXA270 416MHz	64	128	N/A					
P535	Windows Mobile 5.0	PXA270 520MHz	64	256	2Mpixels		2.8"	240x320		802.11b/g
Dell										
Dell Axim X51v-624 MHz	Windows Mobile 5.0	PXA270 624MHz	64	256	N/A			640x480	Intel 2700G	
Axim X51-520 MHz	Windows Mobile 5.0	PXA270 520MHz	64	128	N/A			240x320		802.11b
Axim X51-416 MHz	Windows Mobile 5.0	PXA270 416MHz	64	128	N/A			240x320		
HP										
iPAQ hw6915	Windows Mobile 5.0	PXA270 416MHz	64	128	1.3Mpixels		3"	240x240		
iPAQ hw6920	Windows Mobile 5.0	PXA270 416MHz	64		N/A			240x240		802.11b
iPAQ hw6925	Windows Mobile 5.0	PXA270 416MHz	64		1.3Mpixels	1280x1024		240x240		802.11b
iPAQ hw6940	Windows Mobile 5.0	PXA270 416MHz	64		N/A			240x240		802.11b
iPAQ hw6945	Windows Mobile 5.0	PXA270 416MHz	64		1.3Mpixels	1280x1024		240x240		802.11b
iPAQ hw6500 series	Windows Pocket PC 2003									
iPAQ hx2100 series	Windows Mobile 5.0	PXA270 312MHz	64	64	N/A					
iPAQ hx2400 series	Windows Mobile 5.0	PXA270 520MHz	128	64	N/A					
iPAQ hx2700 series	Windows Mobile 5.0	PXA270 624MHz	128	128	N/A					
iPAQ rx1950 series	Windows Mobile 5.0	S3C2442 300 MHz	32	64	N/A					
Acer										
n311	Windows Mobile 5.0	S3C2440 400 MHz	64	128	N/A			640x480		
E-ten										
ETEN G500	Windows Mobile 5.0	S3C2440 400 MHz	64	128	1.3Mpixels	1280x960	2.8"	240x320		
ETEN M600	Windows Mobile 5.0	S3C2440 400 MHz	64	128	1.3Mpixels	1280x960	1.3"	240x320		802.11b
ETEN M600+	Windows Mobile 5.0	S3C2440 400 MHz	64	256	2Mpixels	1280x960	2.8"	240x320		802.11b
ETEN X500	Windows Mobile 5.0	S3C2442 400 Mhz	64	128	2Mpixels	1600x1200	2.8"	240x320		802.11b/g
HTC										
TyTN	Windows Mobile 5.0	S3C2442 400 MHz	64	128	2Mpixels	+ second cam	2.8"	240x320		802.11b/g
HTC P3600	Windows Mobile 5.0	S3C2442 400 MHz	64	128	2Mpixels	+ second cam	2.8"	240x320		802.11b/g
HTC P4350	Windows Mobile 5.0	TI OMAP850 201MHz	64	128	2Mpixels		2.8"	240x320		802.11b/g

TAB. A.1 – En rouge les motifs d'exclusion des modèles correspondant.

Bibliographie

[1] STUDIERSTUBE, *Klimt - the Open Source 3D Graphics Library for Mobile Devices*, http ://studierstube.icg.tu-graz.ac.at/klimt/

[2] INTEL, *Intel GPP for Intel PCA Application Processors*, http ://www.intel.com/cd/ids/developer/asmo-na/eng/54170.htm

[3] WIKIPEDIA, *OpenGL ES*, http ://fr.wikipedia.org/wiki/OpenGL_ES

[4] THE KHRONOS GROUP, *Open Standards, Royalty Free, Dynamic Media Technologies*, http ://www.khronos.org/

[5] HYBRID GRAPHICS, *Hybrid Graphics*, http ://www.hybrid.fi/

[6] KAMENOV, Boris. Re : windows mobile 5 support. In *Hybrid Graphics Forums* [Forum]. 31.01.2007 http ://forum.hybrid.fi/index.php ?topic=968.0

[7] INTEL, *Intel 2700G Multimedia Accelerator*, ftp ://download.intel.com/pressroom/kits/pxa27x/pb_2700g.pdf

[8] GLUT ES, *GLUT ES - The OpenGL ES Utility Toolkit*, http ://glutes.sourceforge.net/

[9] MICROSOFT, *MSDN Library*, http ://msdn.microsoft.com/