Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Filière Systèmes industriels

Orientation Infotronics

Diplôme 2007

Flavio Tarsetti

Suivi de visage avec un téléphone mobile Windows Mobile

Professeur	Dominique Gabioud
Expert	Dr. Yann Rodriguez

Sion, le 23 novembre 2007

HES-SO Valais

SI	TV	EE	IG	EST
X	X	X	X	

Données du travail de diplôme
Daten der Diplomarbeit

FO.0.2.02.07.DB
DD/18/05/2006

Filière / *Studiengang* : Systèmes industriels

Confidentiel / *Vertraulich* ☐

Etudiant / *Student*	**Année scolaire / *Schuljahr***	**No TD / *Nr. DA***
Flavio Tarsetti	2006/07	SI/2007/11

Proposé par / *vorgeschlagen von*	**Lieu d'exécution / *Ausführungsort***
HES-SO Valais, UIT IDIAP, Dr. Yann Rodriguez, yann.rodriguez@idiap.ch	HES-SO Valais, DSI **Expert / *Experte***

Titre / *Titel*:

Suivi de visage avec un téléphone mobile Windows Mobile

Description / *Beschreibung*:

L'IDIAP (www.idiap.ch), un institut de recherche spécialisé dans les interfaces homme machine multimodales (voix et image en particulier), a développé un algorithme de détection et de suivi de visage(s) dans un flux vidéo appelée FaceTracker ainsi que la librairie Torch (www.torch.ch) qui contient les éléments de base de cet algorithme. L'unité Infotronics a réalisé une application Windows sur la base de cet algorithme et de la libraire Torch.

Le téléphone portable est un équipement informatique multimodal voix-image utilisé par tous et disponible partout en permanence. Ce fait le rend potentiellement attractif pour des services multimodaux même si de très gros problèmes doivent encore être résolus (manque de ressources de calcul, qualité très variable et souvent médiocre de l'information multimodale due au bruit).

Le projet de diplôme a pour but de développer une version de FaceTracker pour un téléphone portable équipé du système d'exploitation Windows Mobile et d'estimer sur la base de ce développement l'adéquation entre les ressources offertes par un téléphone mobile et les ressources demandées par des applications multimodales telles que le suivi de visage.

Objectifs / *Ziele:*

- Mettre en place un environnement de développement pour téléphones mobiles Windows Mobile capable de permettre le développement d'applications multimodales voix-audio.
- Acquérir et documenter le savoir-faire pour le traitement des flux audio et vidéo dans un programme informatique sous Window Mobile : extraction d'une image dans un flux vidéo, extraction d'une séquence audio, affichage d'un flux vidéo généré (en totalité ou en partie) de manière logicielle, envoi d'une séquence audio après traitement vers le haut-parleur.
- Effectuer la phase d'analyse (à l'aide d'UML) pour l'application FaceTracker sur téléphone mobile et, dans la mesure des ressources disponibles, implémenter tout ou partie de cette application.
- Estimer l'adéquation des ressources matérielles des téléphones mobiles Windows Mobile pour des applications multimodales en général et pour l'application FaceTarcker en particulier.

Signature ou visa / *Unterschrift oder Visum*	**Délais / *Termine***
Resp. de l'orientation infotronics	Attribution du thème / Ausgabe des Auftrags: 03.09.2007
Professeur/*Dozent*: Dominique Gabioud	Remise du rapport / Abgabe des Schlussberichts: 23.11.2007
Etudiant/*Student*: Flavio Tarsetti	Exposition publique / Ausstellung Diplomarbeiten: 30.11.2007 Défenses orales / Mündliche Verfechtungen Semaine 49

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats:* .....

Tarsetti, Flavio 2007

Suivi de visage avec un téléphone mobile Windows Mobile

Gesichtsverfolgung mit einem Mobiltelefon mit Windows Mobile

Objectif

Réaliser une application de suivi de visage sur un téléphone mobile équipé du système d'exploitation Windows Mobile. L'application devra acquérir et afficher le flux de donnée vidéo en provenance de la caméra du téléphone mobile. A l'aide des algorithmes de détection de visage fournis par l'IDIAP (institut de recherche spécialisé dans les interfaces homme machine multimodales), qu'il faudra porter sur un système Windows Mobile, l'application devra faire une détection de visage sur ce flux de donnée vidéo.

Résultats

L'application a pu être développée et testée. Elle comprend deux DLL et une interface graphique. L'acquisition/affichage du flux de donnée vidéo en provenance de la caméra du téléphone mobile (avec DirectShow) et les algorithmes de détection de visage ont été respectivement intégrés dans les deux DLL. Quant à l'interface graphique, elle va, d'une part, s'interfacer avec ces mêmes DLL, en dessinant, si besoin, un carré autour du visages et des yeux, et d'autre part, interagir avec l'utilisateur de l'application.

Mots-clés

FaceTracker, face tracking, face recognition, détection de visage, suivi de visage, DirectShow, multimodal application, application multimodale, Windows Mobile

Ziel

Eine Anwendung zur Gesichtserfassung und Verfolgung auf einem Mobiltelefon unter dem Betriebssystem Windows Mobile entwickeln. Die Anwendung fängt dazu den Videodatenstrom der eingebauten Kamera ab und zeigt diesen an. Gleichzeitig führt sie eine Gesichtserfassung und Verfolgung mittels eines aus dem IDIAP (institut de recherche spécialisé dans les interfaces homme machine multimodales) stammenden Algorithmus durch. Zusätzlich muss dieser Algorithmus für die Windows Mobile Plattform portiert werden.

Resultate

Die Anwendung wurde entwickelt und gestestet. Sie umfasst zwei Bibliotheken in Form von DLL's und eine Benutzerschnittstelle. Das Abfangen und Anzeigen des Videodatenstroms der eingebauten Kamera mittels DirectShow sowie die Gesichtserfassung und Verfolgung wurden in die zwei erwähnten Bibliotheken integriert. Die grafische Benutzerschnittstelle erfüllt zwei Aufgaben. Einerseits wird sie in Zusammenarbeit mit den Bibliotheken ein eventuell auf der Anzeige vorhandenes Gesicht mit einem Rahmen versehen und diesen dem Gesicht folgen lassen und andererseits dem Benutzer der Anwendung eine Bedienungsoberfläche anbieten.

Schlüsselworte

Facetracker, Gesichtserfassung, Gesichtsverfolgung, Gesichtserkennung, DirectShow, multimodal application, multimodale Anwendungen, Windows Mobile, sicherheitsrelevante Anwendungen

RAPPORT DU PROJET DE DIPLOME :
SUIVI DE VISAGE AVEC UN TELEPHONE MOBILE WINDOWS MOBILE

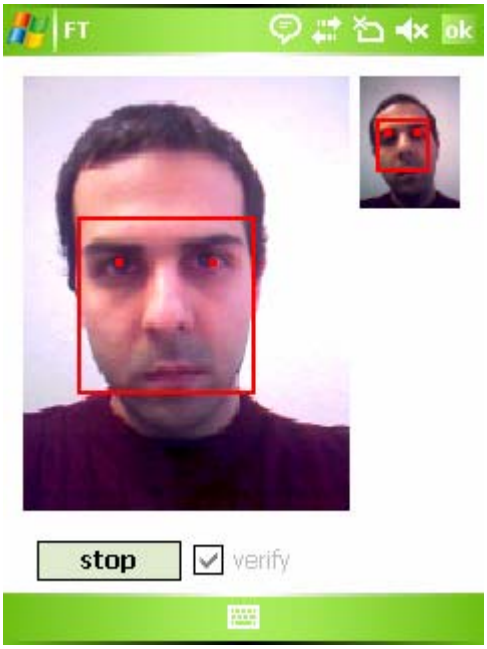


TABLE DES MATIERES

TABLE DES FIGURES

Introduction

Préambule

L'objectif de mon projet de diplôme va être celui de réaliser une application de "suivi de visage" (Face Tracker) sur un téléphone mobile et plus précisément un PDA. Le système d'exploitation pour lequel l'application doit être réalisée est Windows Mobile 5.0 ou Windows Mobile 6.0.

L'objectif général de ce projet est de qualifier et tester la performance d'un téléphone mobile pour des applications multimodales voix images.

Contexte du projet de diplôme

L'IDIAP[1] (www.idiap.ch) est un institut de recherche spécialisé dans les interfaces homme machine multimodales (voix et image en particulier). Cet institut a développé un algorithme de détection et de suivi de visage(s) dans un flux vidéo. C'est dans ce cadre de développement que la librairie Torch (www.torch.ch) a vu le jour. Cette librairie contient en effet les éléments de base de cet algorithme.

L'unité Infotronics de la filière Systèmes Industriels de la HES – SO Valais a déjà réalisé une application Windows sur la base de cet algorithme.

Cadre du projet de diplôme

Dans le cadre du projet de diplôme, il s'agit de :

- Mettre en place un environnement de développement pour téléphones mobiles Windows Mobile capable de permettre le développement d'applications multimodales voix-image.
- comprendre le déploiement sur une cible particulière
- comprendre les outils permettant la réalisation d'un tel projet
- faire une analyse du langage de programmation le plus adéquate à être utilisé pour ce projet

[1] Tiré du site de www.idiap.ch (logo de l'IDIAP): "The **IDIAP Research Institute** is an independent, nonprofit research foundation located in Martigny (Valais, Switzerland). Inaugurated in 1991, and accredited by the State and Federal Governments, IDIAP is affiliated with the "Ecole Polytechnique Fédérale de Lausanne" EPFL and the University of Geneva. IDIAP is committed to leading-edge research and developments in the areas of Speech Processing, Computer Vision, Information Retrieval, Biometric Authentication, Multimodal Interaction and Machine Learning. "

- d'acquérir et documenter le savoir-faire pour le traitement des flux audio et vidéo dans un programme informatique sous Windows Mobile: extraction d'une image dans un flux vidéo, extraction d'une séquence audio, affichage d'un flux vidéo généré (en totalité ou en partie) de manière logicielle, envoi d'une séquence audio après traitement vers le haut-parleur
- d'effectuer la phase d'analyse (à l'aide d'UML) pour l'application FaceTracker (suivi de visage) sur téléphone mobile et, dans la mesure des ressources disponibles, implémenter tout ou partie de cette application.
- d'estimer l'adéquation des ressources matérielles des téléphones mobiles Windows Mobile pour des applications multimodales en général et pour l'application FaceTracker en particulier.

Outils et environnement utilisés

Matériel :

- Un PDA (HTC Touch) équipé du système d'exploitation Windows Mobile 6.0
- Un PDA (HTC P3300) équipé du système d'exploitation Windows Mobile 5.0
- Un PDA (SPV M3100) équipé du système d'exploitation Windows Mobile 5.0
- Un PDA (SPV M2000) équipé du système d'exploitation Windows Mobile 2003
- Une Webcam Logitech RightLight Technology pour les tests

Outils de développement

- L'outil de développement Visual Studio 2005 de Microsoft
- Windows Mobile 5.0 Software Development Kit SDK
- Windows Mobile 6.0 Software Development Kit SDK
- Active Sync 4.5 : Outil de synchronisation PDA ←→ PC
- Framework .NET de Microsoft
- Rhapsody : outil de modélisation UML pour la phase d'analyse

Etude de faisabilité du projet

Il faut rappeler que l'unité Infotronics de la HES-SO Valais avec les librairies Torch de l'IDIAP a déjà pu réaliser une application équivalente pour l'OS Windows XP en utilisant une webcam (nom de l'application : *Sabbuca*). L'application faisait appel à DirectShow pour l'acquisition des données de la webcam. Cette application a été entièrement réalisée avec l'aide des librairies *QT* de Trolltech (site web : http://trolltech.com/).

L'objectif de l'application du projet de diplôme est de faire du "face tracking" sur un PDA.
Pour ce faire, il va s'agir de comprendre les différents moyens à disposition pour mettre en place un tel système et analyser la faisabilité de l'application en fonction de tout cela.

L'application nécessite un environnement de développement qui puisse générer du code ciblant un hardware spécifique, à savoir, le PDA.

Environnement de programmation

L'environnement Visual Studio 2005 permet, par l'intermédiaire du .NET Framework de cibler divers systèmes d'exploitation de Windows.
Pour pouvoir intégrer le système d'exploitation cible, Windows Mobile 5.0 du PDA, Microsoft met à disposition un SDK pour Windows Mobile 5.0 et un autre pour Windows Mobile 6.0 qui, à l'installation, vont s'intégrer à l'environnement Visual Studio 2005.

Synchronisation des fichiers entre les divers OS

La synchronisation et transfert des fichiers sur le système cible sont rendus possibles grâce à l'utilitaire ActiveSync 4.5

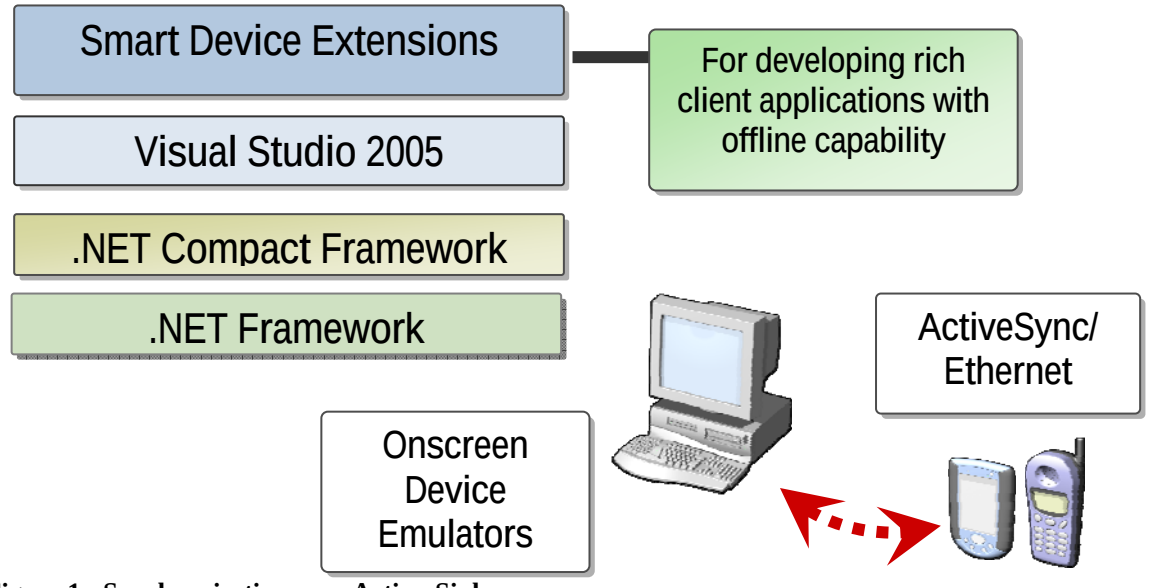


Figure 1 : Synchronisation avec ActivesSink

Librairies de développement

Il existe depuis peu une version de DirectShow ciblant le système Windows Mobile 5.0.

Il n'est pas possible d'employer les librairies *QT* de trolltech pour réaliser l'application car il n'existe actuellement (septembre 2007) pas de versions de QT ciblant le système d'exploitation Windows Embarqué, mais uniquement du Linux Embarqué.

L'application doit donc être en grande partie réalisée à l'aide de DirectShow et de code qui ne dépend pas de librairies *QT.*

Plateforme Windows Mobile

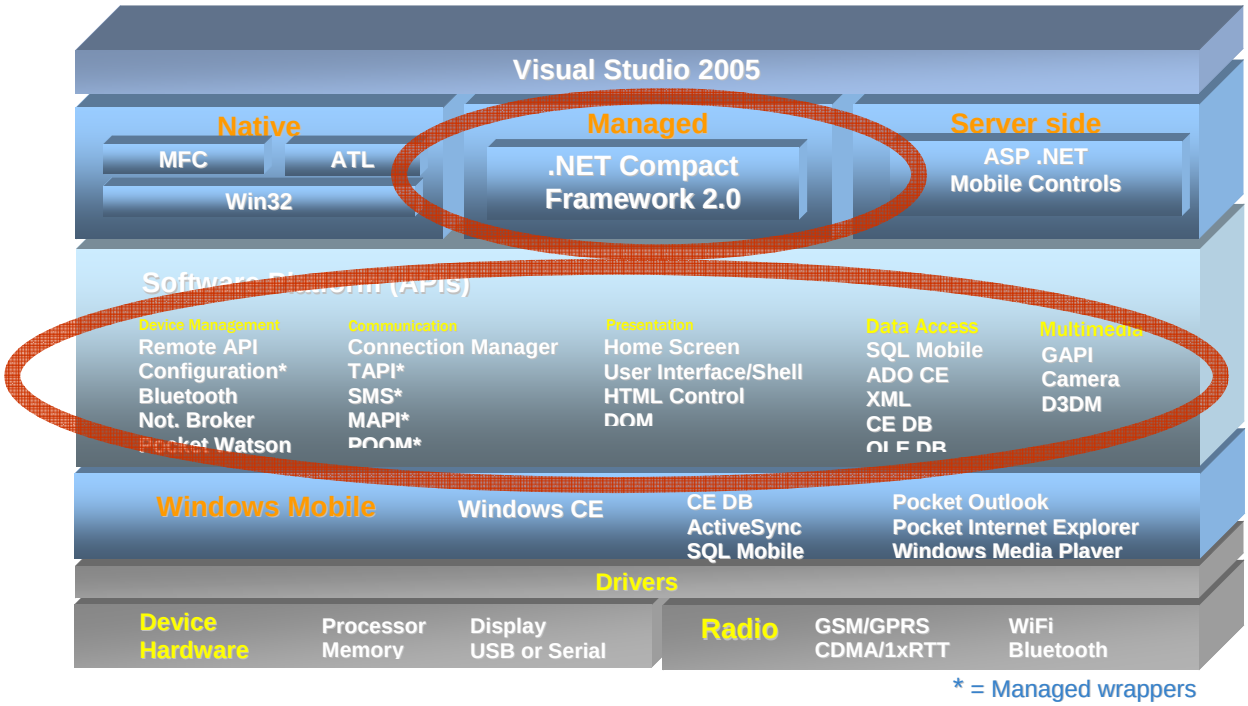


Figure 2 : Plateforme Windows Mobile

Device et plateforme supporté par le compact .NET FrameWork

Device	Platform	.NET CF Version	In ROM
Pocket PC **Pocket PC Phone Edition**	**Pocket PC 2000, Pocket PC 2002** **Windows Mobile 2003 for Pocket PC** **Windows Mobile 2003 for Pocket PC SE** **Windows Mobile Version 5.0 for Pocket PCs**	**1.0** **1.0, 2.0** **1.0, 2.0** **1.0, 2.0**	**-** **1.0 sp1** **1.0 sp2** **1.0 sp3**
Smartphone	**Windows Mobile 2003 for Smartphone** **Windows Mobile 2003 for Smartphone SE** **Windows Mobile Version 5.0 for Smartphone**	**1.0** **1.0** **1.0, 2.0**	**1.0 sp1** **1.0 sp2** **1.0 sp3**
Other Windows CE devices	**Windows CE 4.1** **Windows CE 4.2** **Windows CE 5.0** **Windows CE 6.0**	**1.0** **1.0** **1.0, 2.0**	**1.0** **1.0 sp1** **1.0 sp3**

Figure 3 : Plateforme supporté par le .NET FrameWork

Langage de programmation

Il existe différents langages de programmation sur PDA :

- C++, C# , Java, Visual Basic

Avec la plateforme J2ME de Sun, le langage de programmation Java est la première idée qui surgirait de l'esprit des programmeurs, notamment pour les raisons suivantes:

- il est possible de faire un code aussi petit que possible
- s'adapte à priori pour des machines pas trop puissantes
- Portabilité du code
- Avantage d'un langage interprété pour les plateformes fragiles (exemple: Palm)

Et cependant, ce langage comporte néanmoins plusieurs inconvénients:

- lenteur de Java (plus précisément avec les « threads »)
- gourmand en mémoire

Etant donné que notre application va se baser principalement sur l'utilisation de threads, et qu'il va être question de faire une application "temps réel", il n'est pas possible de se permettre cette lenteur.

De plus, la communauté des développeurs PDA, utilisant le système d'exploitation Windows mobile, est très grande pour les langages suivants : C# et C++.

La différence entre les deux langages est la suivante:

Avec C# nous produisons du code managé, tandis qu'avec C++ nous produisons du code natif. C# est en quelque sorte une version simplifiée de C++ mais avec une syntaxe proche du langage Java.

La communauté Microsoft est plus portée sur le langage C# qu'elle a créée. En effet la plateforme .NET imaginée par Microsoft est totalement adaptée à ce langage de programmation.

La question est de savoir lequel des deux langages choisir:

Il faut utiliser du **code natif** pour de grandes performances, si nous souhaitons accéder par exemple directement au hardware
Il faut utiliser du **code managé** si l'on utilise le .NET Compact Framework dans des applications où l'interface utilisateur joue un grand rôle, et pour les applications qui demandent une grande rapidité de développement.

Et pourquoi ne pas utiliser les avantages des deux langages ?
En effet, le langage C# permet d'aller très vite dans le développement de l'application, mais le langage C++ permet de produire du code natif beaucoup plus efficace.
Etant donné que l'application va avoir accès au hardware du PDA (à savoir, la caméra et le micro), il serait plus judicieux de programmer en C++ tout les accès caméras et calculs, mais de faire l'interface graphique (GUI) avec un contrôle qui supervise le tout en C#.

Dans les pages qui suivent, il sera détaillé très précisément les outils de développement utilisés et les nombreux termes utilisés dans cette étude de faisabilité et d'implémentation du projet.

Description détaillée du matériel et des outils de développement

Description du matériel utilisé

Le PDA actuellement utilisé et disposant du système d'exploitation Windows Mobile 5.0 est le suivant : SPV M3100

Figure 4 : SPVM3100

Un autre PDA sous Windows Mobile 5.0 a pu être testé, il s'agit du HTC P3300

Figure 5 : HTC P3300

Le PDA actuellement utilisé pour Windows Mobile 6.0 est le suivant : HTC Touch

Figure 6 : HTC Touch

Le câble de synchronisation permettant de connecter le PDA au PC (prise USB) est le suivant:

Câble Sync d'Origine pour Htc TyTN / V1605 / SPV M3100 / MTeoR / SPV C700 (RSYS:4:383655)

Figure 7 : câble de synchronisation

Description de l'outil ActiveSync4.5

Microsoft ActiveSync 4.5 permet la synchronisation des appareils mobiles (Pocket PC, Smartphone, etc.) avec l'ordinateur.
Ce logiciel permet ainsi de transférer aisément les photos, fichiers musicaux (mp3, etc.), vidéos, documents, présentations et autres sur le PDA.
Il présente en outre une arborescence de la mémoire de l'appareil mobile connecté au PC, offrant ainsi la possibilité de classer tout les documents et fichiers dans les répertoires et emplacements désirés sur l'appareil mobile.

De plus, ActiveSync 4.5 prend divers types de systèmes d'exploitation en charge, à savoir:

- Windows Mobile 6.0
- Windows Mobile Embedded CE 6
- Windows Mobile 5.0
- Windows Mobile 2003
- Windows Mobile 2003 SE
- Pocket PC 2002
- Smartphone 2002

Il est donc indispensable de l'installer afin de pouvoir transférer les logiciels développés sur la cible qui se trouve être le PDA.

Description de l'outil de développement Visual Studio 2005 de Microsoft

Microsoft Visual Studio 2005 est l'environnement de développement par excellence pour les solutions Microsoft. En effet, il a été réalisé de telle sorte que chaque développeurs individuels puissent imaginer et développer des applications à plusieurs niveaux.
Il regroupe les anciennes versions `eMbedded Visual C++` et les anciennes versions de Visual Studio (2003 et autres).
Il existe plusieurs types de distributions pour Visual Studio 2005, notamment :

- Visual Studio 2005 Team System
- Visual Studio 2005 Professional Edition
- Visual Studio 2005 Tools for the Microsoft Office System
- Visual Studio Standard Edition
- Visual SourceSafe 2005
- Editions Visual Studio Express

La distribution sur laquelle il a été choisi de développer est Visual Studio 2005 Professional Edition.

Cet outil de développement professionnel permet de :
- créer et interagir sur des applications multiniveaux pour Windows
- créer des applications pour le Web

- développer des solutions pour les SmartPhone et Pocket PC
- d'avoir des outils intégrés pour la conception visuelle de base de données, procédures et tables
- de bénéficier d'un afficheur de rapport de base de données intégré
- développer, déboguer et déployer sur une cible particulière des applications multiniveaux
- de travailler avec un débogage XSLT intégré

Pour le projet "Face Tracker", il va donc de soit que cet outil de développement s'avère indispensable car il permet, d'une part de créer des projets orientés pour le monde Microsoft, et également pour le système d'exploitation qui nous intéresse, à savoir : Windows Mobile 5.0 ou 6.0. De plus il permet de déployer directement le programme sur le PDA (Si bien sûr un synchroniseur tel ActiveSync 4.5 est installé sur le PC)

Windows Mobile 5.0 Software Development Kit SDK

Le Software Development Kit (SDK) de Windows Mobile 5.0 est un kit de développement pour les Pocket PC et Smartphone (à installer en fonction de l'appareil souhaité).
Ce SDK étend de facto Visual Studio 2005 de manière à ce qu'il soit possible de développer du software en code managé ou natifs pour la cible Windows Mobile.

Le package SDK comprend :

- des images émulateurs de Windows Mobile 5.0 et des fichiers skins
- des fichiers d'en-têtes (Headers), des librairies, des fichiers IDL et des assemblages de références managées
- des exemples de code
- de la documentation au format CHM
- des outils (Tools), des fichiers de configuration et des certificats d'authentifications de tests

Windows Mobile 6.0 Software Development Kit SDK

Le Software Development Kit (SDK) de Windows Mobile 6.0 est un kit de développement pour les Pocket PC et Smartphone (à installer en fonction de l'appareil souhaité).
Comme pour Windows Mobile 5.0 SDK, ce SDK étend de facto Visual Studio 2005 de manière à ce qu'il soit possible de développer du software en code managé ou natifs pour la cible Windows Mobile.

Le package comprend les mêmes éléments que pour Windows Mobile 5.0 cités plus haut, mais pour le système Windows Mobile 6.0

Pour pouvoir 'installer les deux SDK Windows Mobile 5.0 et 6.0, ces derniers requièrent au minimum les paramètres suivants:

- Système d'exploitation : Windows Server 2003 Service Pack 1 ou Windows XP ou Windows Vista
- Système de développement : Visual Studio 2005 Standard, Professionnel ou la Team Suite Editions
- Le synchroniseur de fichiers entre l'appareil et le téléphone mobile Active Sync 4.0

Il faut savoir que Windows Mobile est basé sur Windows CE et sur le .NET Compact Framework.

L'image ci-dessous démontre quels packages sont important dans le cas d'une installation spécifique de Visual Studio 2005:

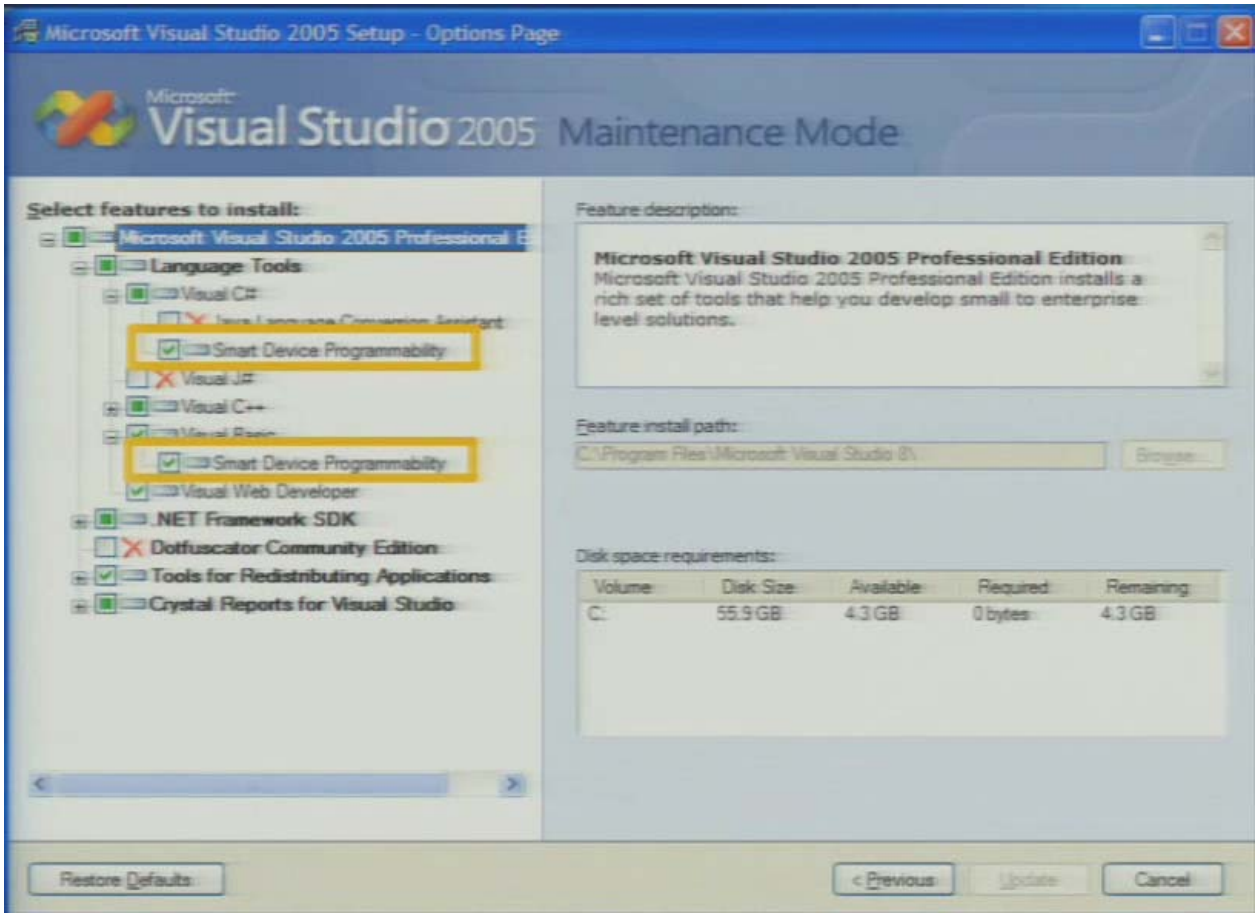


Figure 8 : Installation de Visual Studio 2005 : smart device

L'installation complète simplifiera tout cela et installera tout les composants de Visual Studio 2005.

FrameWork .NET

Ce sous-chapitre sur le FrameWork .NET est largement inspiré du livre « Le langage C++ » des éditions *CAMPUSPRESS* (*Codes_en_Stock*)

La plateforme .NET Framework de Microsoft

Que signifie le .NET Framework ?

Microsoft voulait faire d'Internet une « véritable plateforme de programmation distribuée, permettant aux ordinateurs, systèmes et services de communiquer et de collaborer »[2].
En d'autres termes, cela implique que « dans cet environnement, un code Python, pourra instancier un objet C++ qui hérite d'une classe Eiffel » [2] par exemple.
Au coeur de la plateforme .NET, des programmes écrits dans des langages différents peuvent de cette manière appeler « des fonctionnalités développées dans d'autres langages »[2]. Du côté des développeurs, il suffira d'un minimum d'opérations nécessaires afin de pouvoir adapter les programmes, et du côté des clients, il n'y aura besoin que d'installer le .NET Framework.

Composition du .NET Framework : 3 grandes parties

1) « Le CLR ou runtime (« Common Language Runtime, moteur d'exécution en langage commun ») » [2]

Dans le monde de la pluralité des langages .NET, le code des programmes n'est plus compilé directement en code machine mais en un « langage intermédiaire (Intermediary Language) » nommé IL. Ce code en langage intermédiaire va ensuite « être pris en charge par le runtime(CLR) : celui-ci se charge de la compilation finale au moment de l'installation ou de la première exécution du programme. Le .NET Framework n'étant pas sensible à un langage particulier » [2] nous pouvons aisément « construire les programmes .NET dans de nombreux langages, y compris C++ , Microsoft Visual Basic .NET, JScript sans oublier C#. » [2]
De plus un très grand nombre de langages d'autres éditeurs peuvent également être utiliser, tels « les langages COBOL, Eiffel, Perl, Python ou Smalltalk » [2].

2) Les bibliothèques de classes représentant à eux seuls un incroyable stocks de fonctionnalités dans laquelle les programmeurs des langages .NET peuvent aisément y puiser.

Ces bibliothèques sont semblables à celles disponibles avec C++, sauf qu'elles sont orientées .NET et donc, par là même, accessibles aux langages .NET.

3) ASP.NET, la nouvelle version d'ASP (version Microsoft concurrente de PHP)

[2] Tiré du livre « Le langage C++ » des éditions *CAMPUSPRESS* (*Codes_en_Stock*)

« Le .NET Framework est une technologie essentielle pour le développement ASP.NET. Elle offre les services système de base supportant ASP.NET, ainsi que le développement des Windows Forms, la nouvelle technologie de développement client fournie par .NET.

Le .NET Framework apporte à Windows XP, 2000, NT 4.0, et 98/Me les services système de base pour supporter les technologies .NET . Le Framework est bien sûr intégré aux futures versions des systèmes d'exploitation Windows de type serveur, dont Windows .NET Server fait partie. »[2]

CLR

« Le CLR (ou runtime) fournit un véritable environnement d'exécution pour tout les programmes écrits dans les langages .NET »[2]. Il se charge de gérer l'exécution du code .NET, et en particulier la mémoire et la durée de vie des objets. « Avec C#, le successeur 100% .NET des langages C, C++ ou Java, les programmeurs n'ont plus à effacer de la mémoire les objets dont ils n'ont plus besoin. C'est le programme de récupération de mémoire du CLR (le Garbage Collector) qui se charge de ces opérations. En plus de ces services de gestion le runtime permet aux développeurs d'étendre le débuggage, la gestion des exceptions et l'implémentation de l'héritage à un environnement multilangage.

Pour que (notre) programme puisse bénéficier de l'environnement d'exécution du CLR, (nous devons) bien sûr faire appel à un compilateur compatible avec le runtime, par exemple Visual C++, Visual Basic, C#, JScript, ou l'un des nombreux compilateurs tiers tels que Perl ou Cobol. Tous les fichiers requis pour développer en Visual Basic, C++ ou C# (sont) intallés avec le SDK. »[2]

Les fichiers du runtime et les compilateurs se trouvent dans le répertoire *monRepertoire\Windows\Microsoft.NET\Framework\versionX*

Les fichiers les plus importants du SDK

« Mscorblib.dll. Cette bibliothèque fournit l'espace de noms System que tout programme .NET utilise. Les langages Visual Basic et C# font implicitement référence à cette bibliothèque, mais dans les extensions managées pour programme C++, (il faut) introduire la directive de préprocesseur `#using <mscorlib.dll>`
`Vbc.exe` . Le compilateur Visual Basic .NET.
`Cl.exe` . Le compilateur C++ .NET.
`Csc.exe` . Le compilateur C#.

Ces compilateurs doivent aussi suivre la CLS (Common Language Specification), qui décrit un sous-ensemble des types de données supportées par le runtime et communs à tous les langages utilisés dans .NET. »[2]

[2] Tiré du livre « `Le langage C++` » des éditions *CAMPUSPRESS* (*Codes_en_Stock*)

Windows Mobile les différents systèmes d'exploitations

Windows Mobile en est à ce jour (5 juin 2007) à la version 6.0. Windows Mobile 6.0 étant assez proche de Windows Mobile 5.0, il n'a donc pas été représenté sur le graphique ci-dessous. Toutefois il faut remarquer que pour programmer sur un système équipé de Windows Mobile 6.0, il faut être équipé du SDK Windows Mobile 6.0.

Windows Mobile, une vue générale sur les versions

Figure 9 : Vue générale sur les versions de Windows Mobile

Il existe différents types de versions comme nous pouvons le remarquer :

L'OS des systèmes embarqués chez Microsoft est Windows CE. Il existe donc dès que nous avons un périphérique embarqué.

Pour plusieurs raisons, dont la raison marketing, il y a eu une déclinaison qui a été faite de cet OS Windows CE: Pocket PC et Smartphone.

Un Pocket PC représente un PDA qui dispose d'un écran tactile. Il est possible de naviguer à travers les menus avec l'aide d'un petit stylet.

Un Smartphone lui est équivalent au Pocket PC, mis à part le fait qu'il ne dispose pas d'un écran tactile, mais possède néanmoins deux boutons latéraux, à droite et à gauche de l'appareil, qui permettent de naviguer dans les menus.

DirectX

Introduction

Apparue en 1995, DirectX portait le nom de « The Game SDK » car développé essentiellement pour le monde du jeu sur ordinateur.
DirectX, développé par Microsoft, est une suite d'API fournissant les principales bibliothèques permettant de faire du traitement audio/vidéo (carte vidéo, carte son,…) ainsi que sur les périphériques d'entrées/sorties (souris, clavier, carte réseau,…).
Faisant parti du système d'exploitation « Windows » il va donc être possible d'accéder au matériel d'un ordinateur exploitant ainsi les capacités de ceux-ci.

La famille DirectX, car on peut véritablement parlé de famille est constituée d'un ensemble de modules dont l'indépendance peut varier.

Famille DirectX :

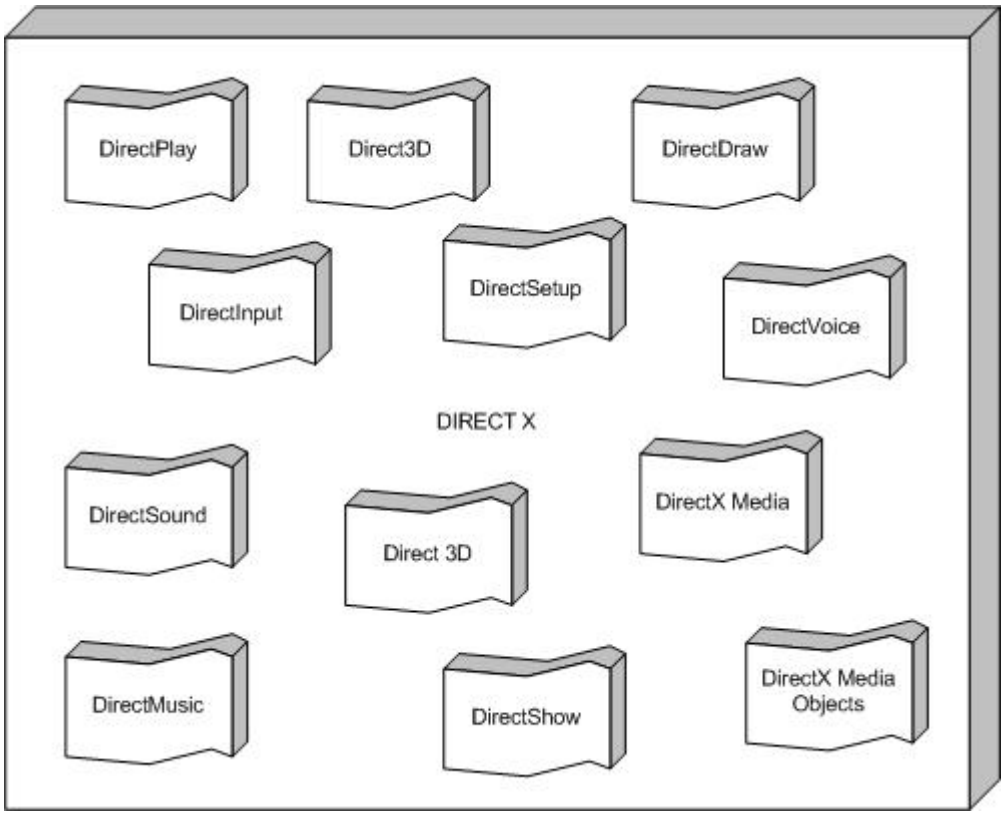


Figure 10 : la famille DirectX

Les modules de la famille DirectX sont les suivants :

- Direct3D : gère l'affichage 3D (ajouté à partir de la version 2)
- DirectDraw : gère l'affichage 2D Raster (intégré à Direct3D depuis la version 8)
- DirectInput : gère les périphériques d'entrée (clavier, souris, joystick, volant et autres controleurs de jeu)
- DirectPlay : gère les échanges de données par réseau (local ou Internet)
- DirectSound : gère les sons (remplacé par DirectSound3D depuis la version 3)
- DirectMusic : API de DirectSound
- DirectVoice : gère les échanges vocaux en direct
- DirectSetup : gère l'installation de composants DirectX (ce n'est pas vraiment une API)
- DirectShow : gère l'affichage et la capture de vidéo
- DirectX Media : framework incluant DirectAnimation, DirectShow, DirectX Video Acceleration, Direct3D Retained Mode et DirectX Transform pour l'animation, la lecture multimédia et les applications de streaming, la 3D, et l'interactivité
- DirectX Media Objects : gère le support des objets streaming tels que les encodeurs, les décodeurs et les effets

Figure 11 : Image tirée du site *: « http://fr.wikipedia.org/wiki/DirectX »*

Il est toutefois à remarquer que le module DirectShow ne fait plus véritablement partie de la famille DirectX. En effet il fait désormais partie du PlatformSDK.

DirectX et DirectShow sur PC et le "Plateform SDK"

Pour pouvoir programmer avec DirectX, il va falloir installer le SDK de DirectX que l'on trouve aisément sur le site officiel de Microsoft.
Cependant, pour programmer avec DirectShow, il ne suffit plus d'installer le SDK de DirectX. En effet, si DirectShow faisait autrefois partie de DirectX, ce n'est désormais plus le cas. Ainsi, pour pouvoir programmer avec DirectShow sur un PC normal, il va falloir installer séparément le Platform SDK qui contient également DirectShow et l'éditeur de graphe de filtres, le fameux "*GraphEdit*".

Directshow (souvent abbrégé par DS ou DShow) appelé aussi *Quartz* est un framework multimédia et une API construite par Microsoft pour les développeurs software afin de leur permettre de réaliser diverses opérations sur les fichiers multimédia. Il remplace l'ancienne technologie de Microsoft "Video for Windows". Basé sur le Microsoft Windows Component Object Model (COM) framework, DirectShow fournit une interface commune pour le multimédia au travers de plusieurs langages de programmation de Microsoft. Il représente également un filtre extensible permettant de rendre ou enregistrer différents fichiers multimédias.
DirectShow contient des plugins DirectX pour le traitement du signal audio et pour "DirectX Video Acceleration" se référant aux playback vidéo accélérer.

La plupart des applications Windows relié au monde vidéo utilisent DirectShow. L'exemple bien connu en est le player de Microsoft: "*Windows Media Player.*"
Le concurrent direct de DirectShow sur les ordinateurs Apple est le framework QuickTime.
L'avenir de DirectShow est incertain car, Microsoft prévoit bientôt un remplacement de ce dernier par le "*Media Foundation*"

L'utilitaire GraphEdit permet de visualiser une application DirectShow. Cependant, il faut comprendre que GraphEdit ne permet pas de réaliser une application complète et complexe. En effet, l'objectif de GraphEdit est uniquement de familiariser le développeur au monde de DirectShow.
Une application DirectShow avec cet utilitaire se présente de la façon suivante:

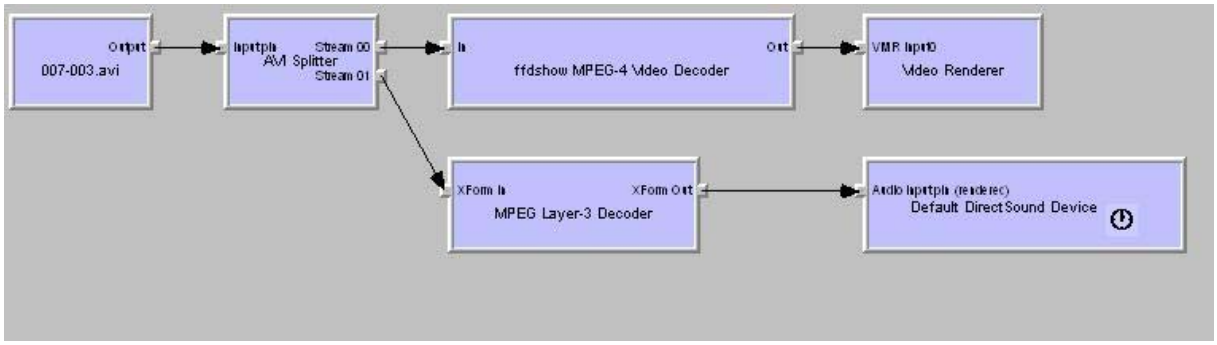


Figure 12 : GraphEdit : Graphe defiltres DirectShow

Explication des éléments du graphe

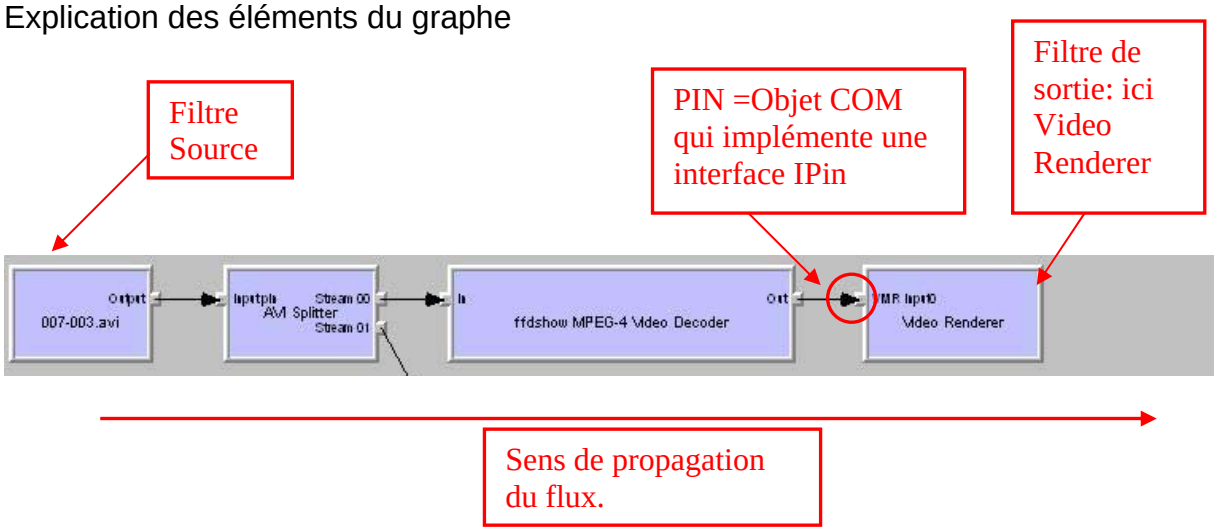


Figure 13 : GraphEdit : Graphe de filtres DirectShow

Un exemple d'application sera développer dans la suite de ce document.
Nous remarquons tout de même que les filtres et les pins sont des objets de type COM.
Remarque : les objets COM permettent la communication interprocess

L'application Face Tracking

Explications de la philosophie du programme

L'objectif comme expliqué, est de réaliser une application qui s'occupe de détecter un visage (face tracking) sur un flux vidéo ainsi que de dessiner l'amplitude de la voix.

A l'étape actuelle du projet, la partie "face tracker" (détection de visage) sera plus traiter que la partie "voice tracker"(ou plutôt devrions-nous dire « realtime audio envelope visualisation » car il ne s'agit que de dessiner l'enveloppe de la voix).

Face Tracker: Use Case Diagram

En d'autres termes, si nous représentons l'application sous la forme d'un diagramme de cas d'utilisation, nous remarquons que ce dernier est très simple :

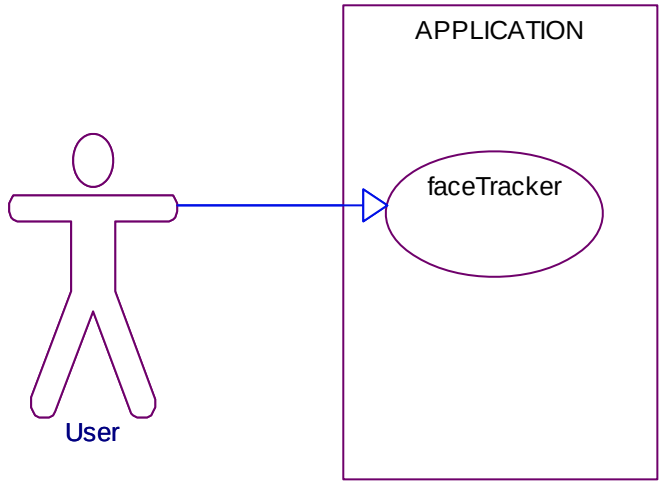


Figure 14 : "Face Tracker" Use Case Diagram

En effet, comme nous pouvons le voir, le cas d'utilisation se limite à l'utilisateur qui ne fait que présenter son visage à l'application qui se charge de faire tout le traitement de détection faciale.

Face Tracker: Collaboration Diagram

Le diagramme de collaboration général ci-dessous montre, lui, une vue avec les interactions entre l'utilisateur et l'application

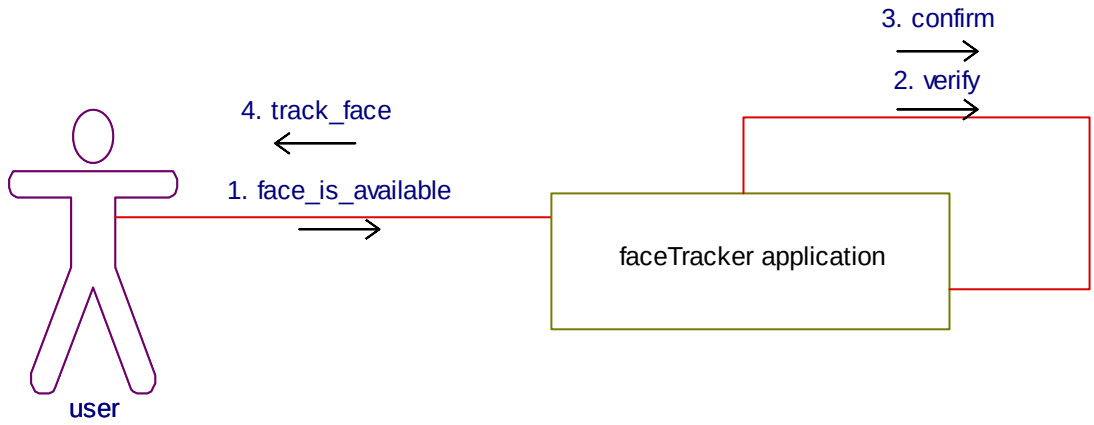


Figure 15 : «Face Tracker » collaboration diagram

Structure du programme

Explication de la première version de la structure:

L'idée de départ de l'application était de faire en sorte qu'elle traite dans un seul et unique thread : le traitement audio et vidéo et l'affichage sur l'écran du PDA.
La figure ci-dessous en illustre le principe :

Figure 16 : 1ère version de la philosophie du programme « Face Tracker »

Toutefois ce programme va entraîner les nombreux problèmes suivants:

- programme non modulable
- affichage lent dû au temps de traitement de l'image
- impossibilité de séparer les tâches
- difficulté de débugger correctement
- l'obligation de traiter chaque image implique des délais beaucoup trop importants et peut amener au "crash" de l'application

C'est pourquoi cette version a été délaissée au profit d'une deuxième version.

Explication de la deuxième version de la structure:

L'idée est de séparer les divers traitements de l'image et l'affichage du flux vidéo. En d'autres termes, nous arrivons à l'idée d'une application découpée en plusieurs parties que l'on pourra par la suite intégrer dans des threads.

Si nous essayons de nous concentrer sur l'application finale, nous pouvons clairement scinder cette dernière en trois parties bien distinctes :

a) l'acquisition vidéo
b) la détection de visage
c) le GUI (« Graphical User Interface » ou interface graphique)

En effet si nous observons cela comme un système de « puzzle » où tout s'interfacera de façon plus logique, nous pouvons avoir la vue globale suivante de l'application :

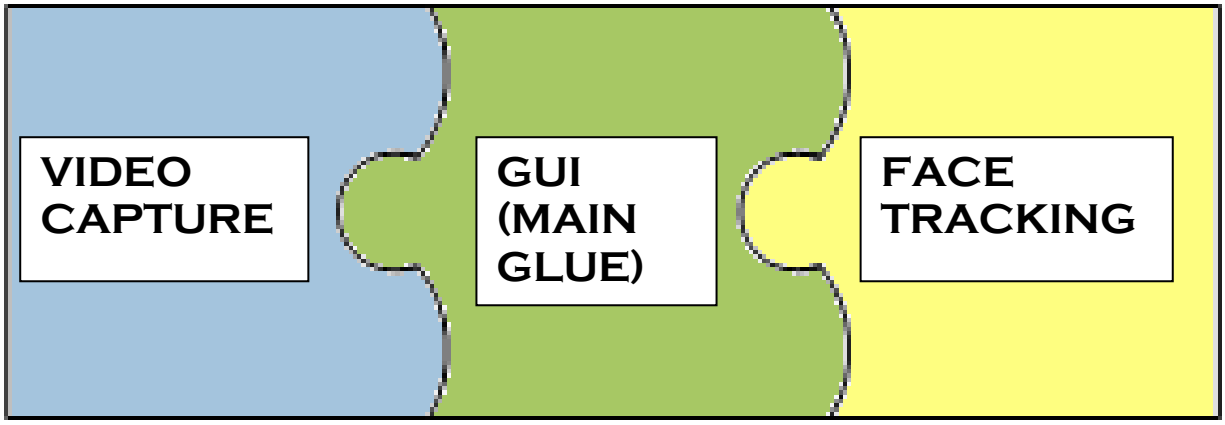
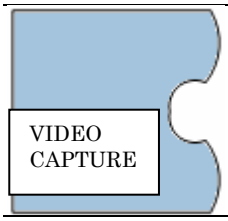


Figure 17 :2ème version de la structure du programme « Face Tracker »(puzzle structure)

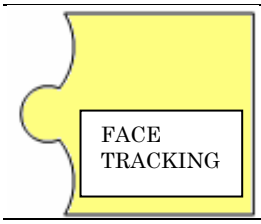
Où chacun des éléments représente :



La capture où acquisition vidéo. :

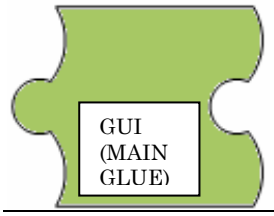
Ce bloc symbolise toute la partie qui va se charger de mettre en place DirectShow dans l'application. Il comprend notamment les éléments tels que la détection de la caméra du PDA (dans le cas où il y a plus d'une caméra sur le PDA), et du driver de la caméra. L'objectif de cette partie est d'avoir un contrôle total du flux vidéo.
De ce flux vidéo, il va s'agir de le maîtriser totalement, à savoir : la taille de la vidéo, la possibilité de le lancer (« play »), le stopper(« stop »), faire un « pause », mais

également de prendre une photo, où plutôt de prendre trame(« frame ») du flux vidéo. (Cette « frame » sera utilisée dans le bloc « Face Tracking »)



La détection de visage (Face Tracking) : Image Processing

Ce bloc symbolise toute la partie qui va se charger de mettre en place l'interfaçage entre le détecteur de visage (fourni par les algorithmes de la librairie Torch de l'IDIAP) et l'application. Elle va contenir notamment un véritable « wrapper » ou « enveloppe » autour de la librairie Torch une fois portée sur Windows Mobile. Ce bloc va prendre une image en entrée (un fichier bitmap (BMP) 24 bits) et lancer le détecteur de visage de Torch sur l'image afin de détecter un visage sur l'image. Si un visage a été détecté, il va alors être possible d'extraire des données intéressantes de la détection, à savoir, les coordonnées en x et y (axe horizontale et verticale) des pixels de l'image, où se trouvent les yeux, et les coordonnées qui détermine le rectangle du visage (coordonnées (x ; y) du coin supérieur du rectangle et largeur et hauteur). L'algorithme va également retourner un « score » qui va permettre de déterminer la fiabilité de la détection du visage dans le cas d'une application d'authentification de personnes.



Le GUI (Graphical User Interface ou «interface graphique ») maître de l'application

Ce bloc symbolise toute la partie qui va se charger de mettre en place la logique de l'application. En effet elle garantit le bon interfaçage entre les divers pièces du « puzzle », soit donc dans ce cas présent : les blocs « Video Capture » et « Face Tracking ».
Ce bloc va également offrir une interface graphique permettant d'interagir avec l'utilisateur de l'application.
Véritable pièce maîtresse de l'application, ce bloc va notamment pouvoir lancer les « Threads » (ou « processus ») dont il a besoin pour le fonctionnement de l'application.
A cette fin, il agit véritablement comme une couche d'assemblage des fonctions d'autres blocs de telles sortes à avoir une cohérence dans l'application finale.

Il s'agit toutefois de noter qu'il ne s'agit pas d'un bloc qui doit rester figé, dans ce sens où il doit pouvoir être extensible dans l'avenir avec d'autres fonctionnalités tels que par exemple : la détection vocale (Voice Processing), etc…

Exemple d'interfaçage futur de l'application pour de la détection vocale (Voice Processing).

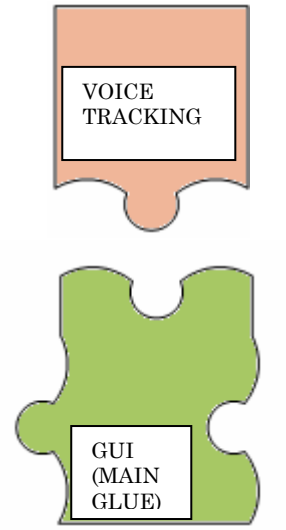


Figure 18 : insertion d'une nouvelle pièce de puzzle dans la philosophie du programme

Mise en relation des blocs

La question est maintenant de déterminer l'interaction entre les divers blocs. En effet, sachant que nous disposons maintenant des éléments nécessaires à la mise sur place d'une telle application, il nous faudrait déterminer la façon dont ces derniers vont interagir entre eux.

En effet, nous partons de la situation initiale suivante :

Les blocs sont développés et prêt à être utilisés

Figure 19 : Pièces de puzzle sans liens les uns avec les autres

En ajoutant, une couche d'abstraction ou d'interfaçage, à l'image d'une pâte collante telle l'image ci-dessous :

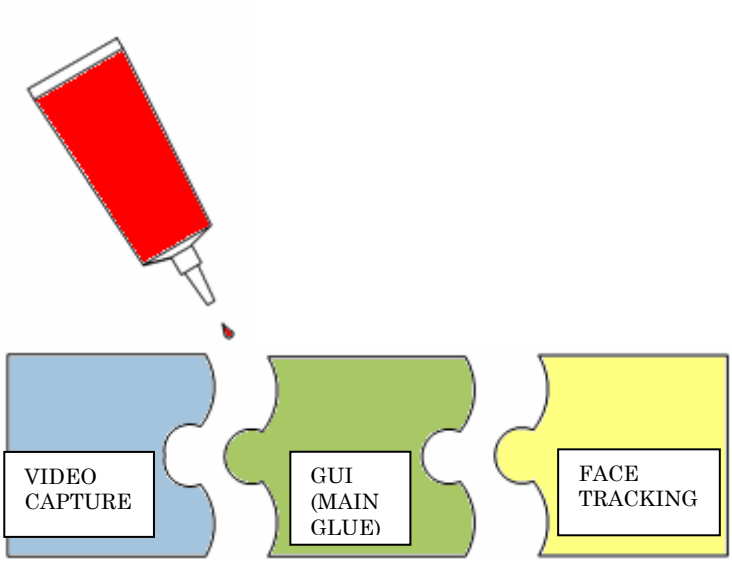


Figure 20 : Insertion d'un lien qui va unir les pièces de puzzle entre elles

L'objectif est d'arriver à la situation finale suivante :

Les blocs (ou « puzzle » dans la figure ci-dessous) sont totalement indépendants et uniquement le GUI, pièce maîtresse du système, pourra discuter à souhait avec l'un ou l'autre des blocs.

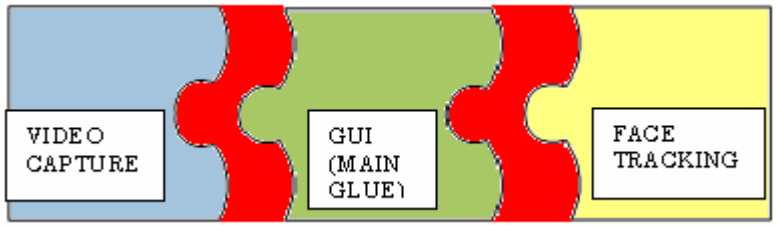
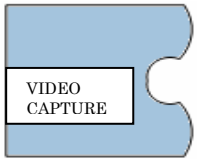


Figure 21 : Un lien a été créé entre les pièces

Dans cette optique, il faut considérer le projet à la fois dans son contexte général tout en ayant une vue spécifique à chaque bloc :

Sans entrer dans les détails pointus des « pièces » de l'application, nous pouvons dire sans trop de peine que le bloc :

A)



Doit être :

- capable de prendre le contrôle de la caméra du PDA (driver,…)
- permettre de visualiser un flux vidéo (play , pause, stop,…)
- offrir la possibilité de prendre une image du flux vidéo et de l'enregistrer sur le PDA
- le plus rapide possible
- capable d'exposer aisément les fonctions au GUI

Nous pouvons donc dire que, ce bloc doit donc être codé en C++ afin de remplir les conditions de performances et de rapidité d'exécution du bloc.
En effet, nous ne pouvons nous permettre de perdre du temps avec du code managé (C#) car il sera compilé en « langage intermédiaire » par le biais de la CLR. De plus ici avec DirectShow nous allons directement toucher le hardware de la caméra, il nous faut donc passer par du code natif beaucoup plus apte à permettre une bonne interaction avec le hardware.

B)

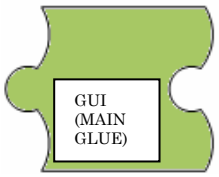
Doit être :

- capable de prendre une image en entrée
- capable d'interagir efficacement avec la librairie Torch de l'IDIAP (soit donc de détecter s'il y a un visage sur une image, et d'en ressortir les coordonnées correspondant à la position du visage sur la photo, de trouver la position des yeux,…). La librairie Torch est écrite en C++
- en mesure d'exposer correctement ses fonctions au bloc GUI
- être le plus efficace possible

Dans la mesure où ce bloc va englober le code de la librairie de l'IDIAP basée sur Torch, nous pouvons supposer que du code va devoir être adapté ou modifier légèrement dans cette librairie de détection de visage. Afin de rester cohérent avec le code déjà mis en place dans la librairie, une classe « enveloppe » ou « wrapper »

va devoir être développée dans le même langage de cette librairie, soit donc, en C++. Cela offre de plus l'avantage d'être beaucoup plus performant dans les temps de traitement des images et des éventuels accès à effectuer.

C)



Doit être :

- capable de synchroniser les différents blocs (capture vidéo, face tracking, …)
- apte à mettre sur place un GUI
- à même de lancer la capture de la caméra et la fenêtre de « preview »
- à même de lancer la détection de visage
- capable de récupérer les informations de la détection de visage
- capable de dessiner sur le flux vidéo 3 carrés si un visage à été détecter (1 carré pour symboliser le visage et deux autres pour indiquer les yeux)

Cette pièce de notre « puzzle » étant la pièce maîtresse, nous pouvons très facilement nous rendre compte qu'elle va se charger de lancer les fonctionnalités des autres pièces du « puzzle » dans des threads (ou « processus ») séparés, et lancer les fonctionnalités permettant de dessiner sur une vidéo.

Ne nécessitant pas une importante rapidité de calculs mais plutôt une excellente gestion à la fois des divers threads démarrés que d'une interface graphique, cette partie va donc être développée en C#.

Assemblage des blocs :

N'oublions pas que nous sommes dans un environnement .NET. Et en effet, la philosophie de .NET est de pouvoir programmer dans une multitude de langages différents tout en permettant d'assembler les divers programmes qui auraient pu être créé.
Nous avons ici deux langages qui ont été utilisés, à savoir : le C++ et le C#

Le C++ permet de développé du code natif et donc « non managé » (ou « unmanaged » code).
Le C# utilisant automatiquement la CLR nous avons alors du code « managé » (ou « managed » code)

Afin de faire communiquer les deux langages ensembles, et dans le but d'exposer les fonctions « proprement » au GUI, il serait intéressant de pouvoir exporter uniquement les fonctions dont nous avons besoin. Une bonne manière serait de

déployer le code C++ dans une DLL et d'exporter les fonctions de ce code afin de pouvoir importer ces mêmes fonctions dans le code C#.
Nous créons ainsi, virtuellement une séparation entre les deux mondes.
La capture vidéo et le face tracking vont donc constitués deux DLL en C++ séparés qui vont exportés des fonctions qui seront importés par le GUI en C#.

Cette manière de fonctionner semble d'ailleurs être celle proposée pour DirectShow sur Windows Mobile par MSDN à l'adresse http://msdn2.microsoft.com/en-us/library/aa454909.aspx#working_with_multimedia_topic7 (tels que consutlés en date du 18 novembre 2007)
« A good general approach is to implement the user interface, business logic, database access, and basic camera interaction by using managed code.(…). If you want to access DirectShow from managed code, (a) solution is to package the use of DirectShow in a native DLL that your managed code calls by using platform invoke. This is probably a good solution for functionality. »

Nous arrivons enfin donc à la version finale suivante de notre programme :

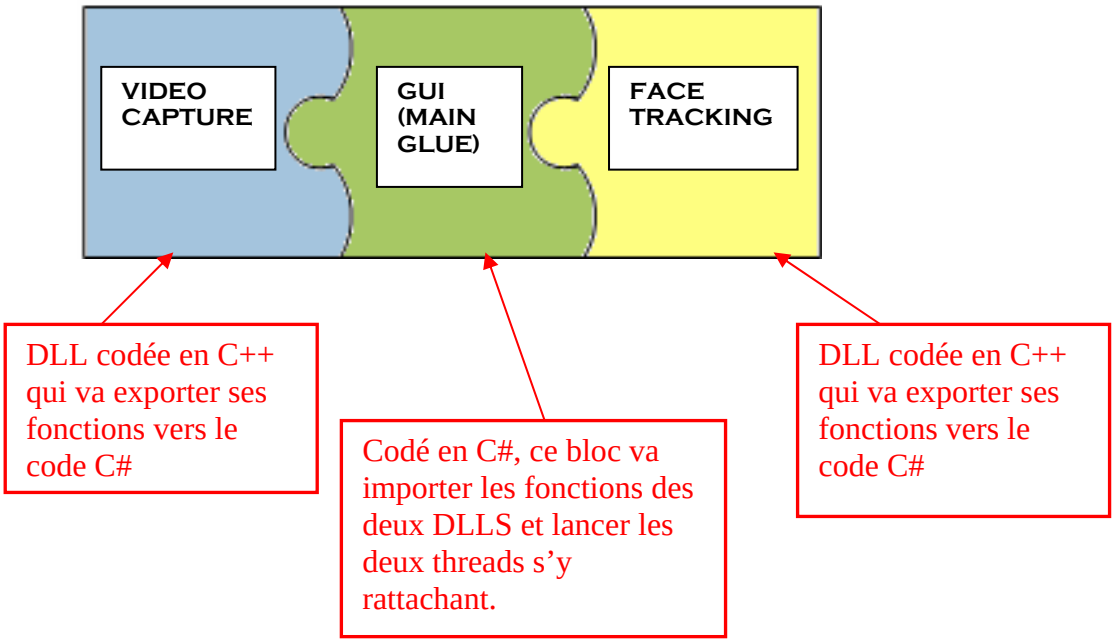


Figure 22 : Présentation de la philosophie du programme tel qu'il va être développé

Nous retrouvons les mêmes éléments dans notre projet :

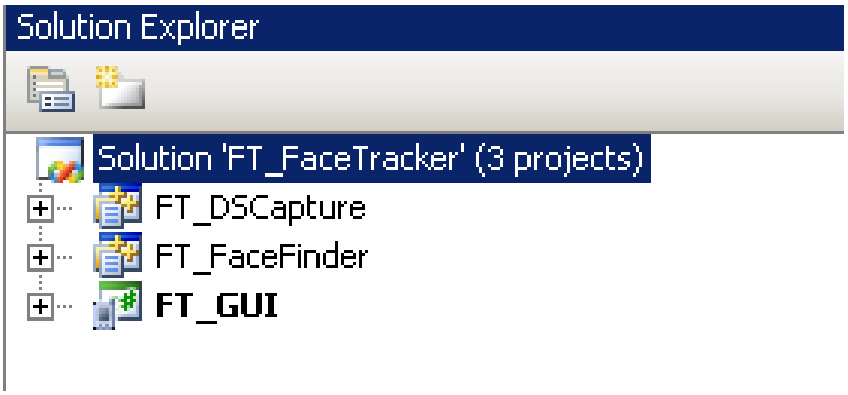


Figure 23 : Printscreen de l'explorateur de Solution

Où :

- FT_DSCapture représente le projet pour la DLL du « puzzle Video Capture »
- FT_FaceFinder représente le projet pour la DLL du « puzzle Face Tracking »
- FT_GUI représente le projet pour le « puzzle GUI »

Vision plus détaillée de cette deuxième version de la structure découpée par threads:

L'idée était de séparer le traitement et l'affichage. En d'autres termes, nous arrivons à l'idée d'une application découpée en plusieurs threads.

La figure ci-dessous en illustre le principe:

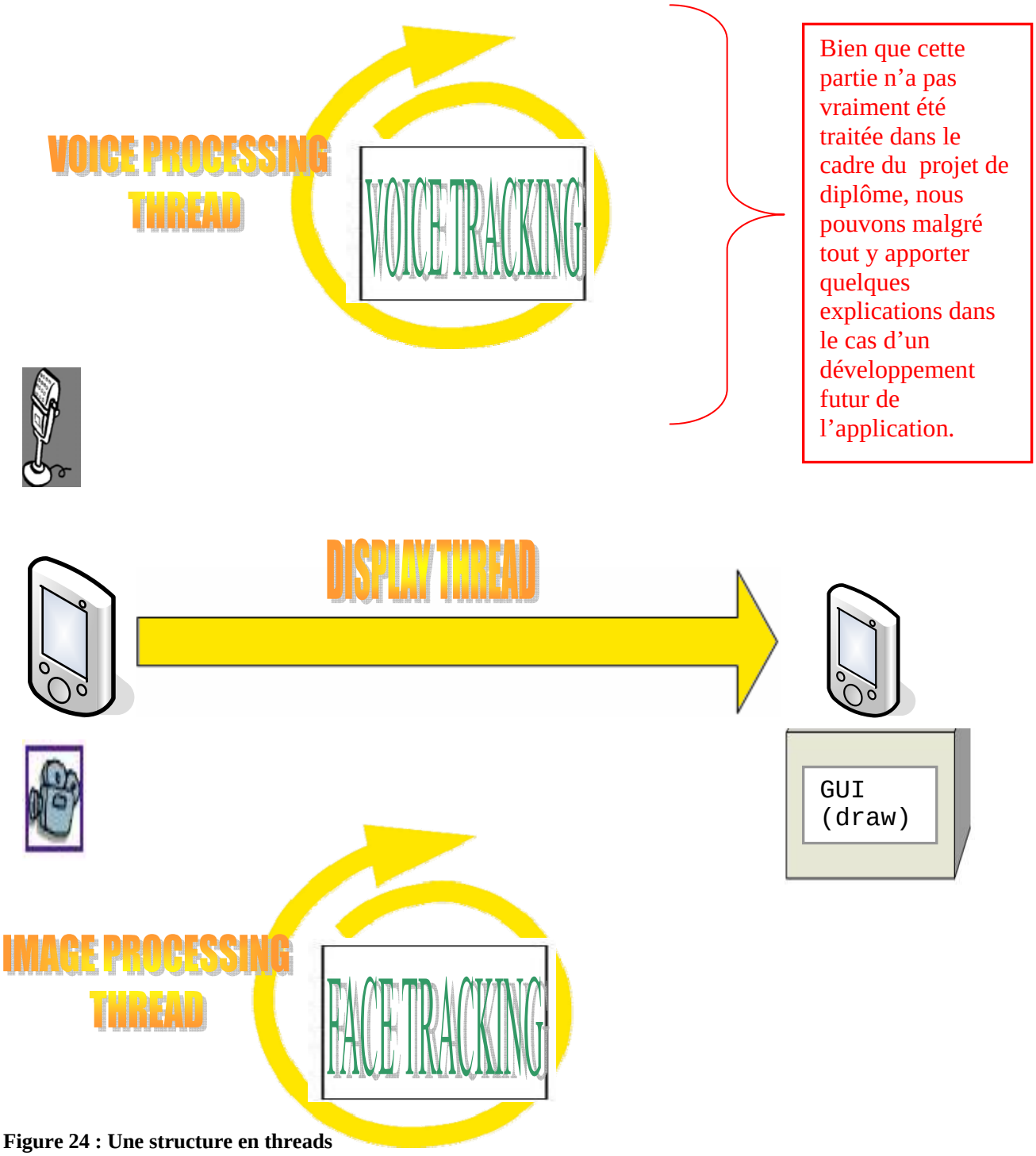


Figure 24 : Une structure en threads

Explications :

Le programme va se structurer de la façon suivante :

3 threads (ou processus) vont être nécessaires afin d'accomplir la tâche souhaitée :

Le thread principal va s'occuper d'afficher le streaming vidéo de la caméra. Sa seule et unique fonction va être de capturer l'image de la caméra et de la rendre dans une fenêtre créé par l'application.

Les deux autres threads vont s'occuper de mettre à jour les coordonnées de l'image et du son.

Le thread pour le son va s'occuper de prendre un sample et de l'échantillonner périodiquement de sorte à prélever la valeur actuelle où il se trouve. Ensuite il mettra à jour les coordonnées des points de l'enveloppe du son sur l'image.
Le thread pour le face tracker va lui s'occuper de mettre à jour les coordonnées du rectangle pour le visage et les coordonnées des ellipses pour les yeux.

Le GUI va se charger de redessiner par dessus chaque image : le rectangle du visage, les ellipses pour les yeux et l'enveloppe du son.
Ces données pour redessiner sont régulièrement mises à jour par les deux autres threads.

En comparaison avec la première version du programme, cette version se trouve nettement améliorée pour les raisons suivantes:

- les différentes tâches sont séparées amenant à une application plus modulable
- le processus de traitement de l'image a son propre thread
- le processus de traitement du son a son propre thread
- le processus d'affichage a son propre thread
- le débuggage de l'application va se trouver d'autant plus facilitée
- il n'y a pas d'obligation de traiter chaque image

Cette phase d'analyse amène donc à choisir cette deuxième version proposée pour l'application plutôt que la première pour les raisons citées ci-dessus.

Explication détaillée du thread de détection faciale:

Le diagramme ci-dessous en illustre le principe :

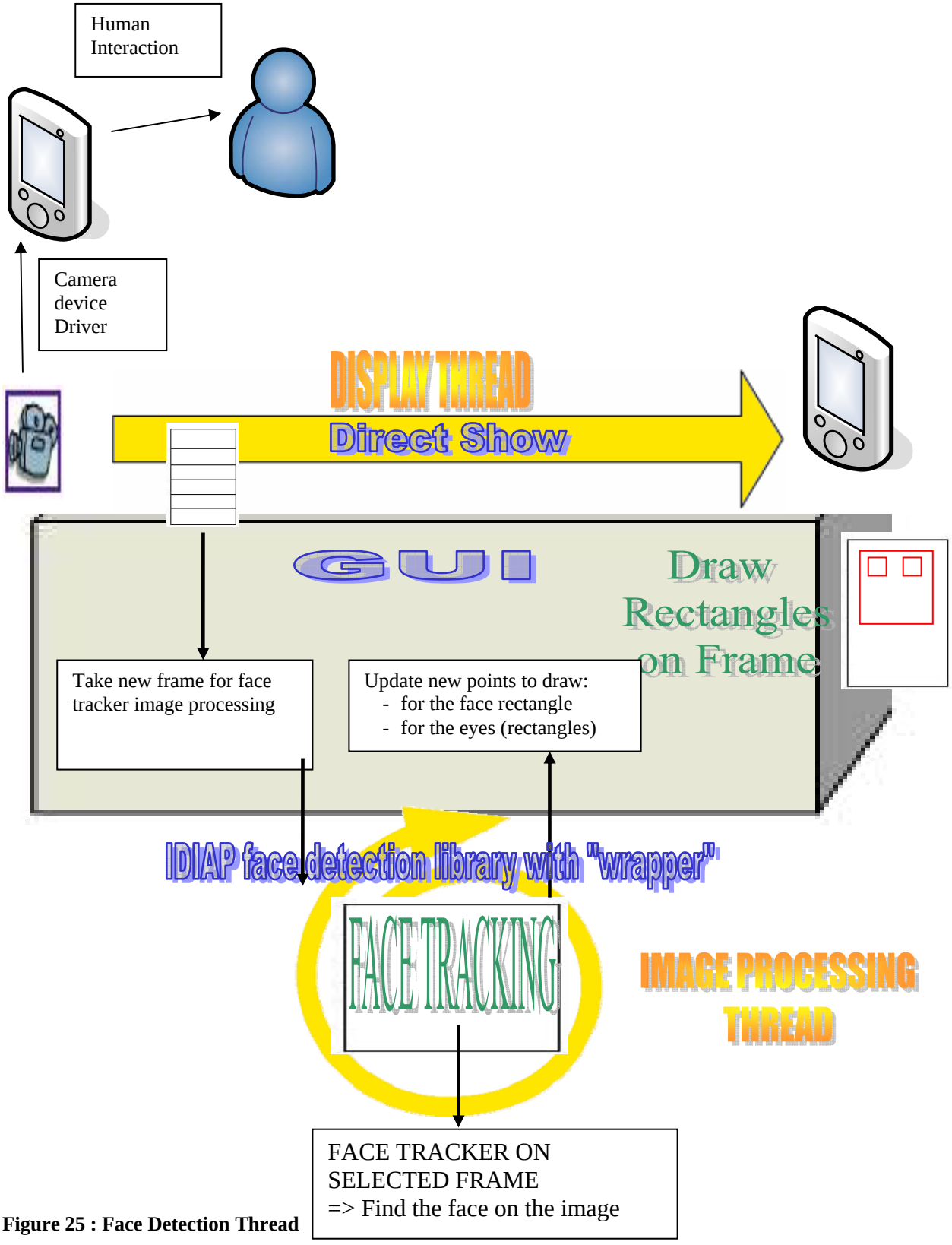


Figure 25 : Face Detection Thread

Explications

Le « display thread » est le processus d'affichage ; il va se charger d'afficher les images les unes après les autres sur l'output du système qui est l'application avec le « stream vidéo » de la caméra. Il va également offrir la possibilité au GUI de lui « voler » une image du flux vidéo et de l'enregistrer comme un fichier « *.bmp » sur le PDA.

Le GUI va se charger de prendre une frame du flux pour la détection de visage en lançant la fonction d'enregistrement au format « *.bmp » du « display thread ». Il va ensuite passer ce fichier « *.bmp » au « face tracking thread » qui va lui retourner un booléen ainsi que des entiers. Ce booléen indique si oui ou non un visage a été trouvé et les entiers donnent la position du visage et des yeux sur l'image.
Ce GUI va se charger de contrôler la valeur du booléen. Si la valeur du booléen est '1' alors un visage a été détecté, sinon il vaudra '0' et aucun visage n'a été trouvé par le détecteur de visage.
Si un visage a été trouvé, le GUI va se charger de dessiner et mettre à jour sur le flux vidéo les coordonnées des carrés représentant respectivement le contour du visage et les yeux de la personne détectée.
Si aucun visage n'a été détecté, bien entendu, aucun carré ne sera affiché à l'écran.
Toutefois ce thread ne se charge pas de trouver le visage. Il ne se charge que de dessiner et rien de plus!

Le « thread de détection facial » se charge de faire le traitement d'image à l'aide de l'algorithme de détection de visage fourni par la librairie Torch de l'IDIAP.

Vision idéale :
L'idéal serait que cette opération de traitement d'image se fasse sur une « frame » parmi N. L'objectif serait d'avoir une fréquence d'échantillonnement du flux vidéo se trouvant entre 50 et 100 [Hz] afin de faire du "face tracking" en moyenne une image toute les 10 [ms].

Vision réelle :
Ces paramètres ne peuvent pas être appliquées dans le cadre du projet de diplôme. Comme nous le verrons plus loin, la détection de visage prend en général entre 1 et 2 secondes.
Dans ce cas, nous allons changer la philosophie qui est celle de régler la fréquence d'échantillonnement du flux vidéo et laisser au contraire l'appareil régler cela. En effet, dès que le système a finit avec le détecteur de visage et le dessin sur le graphique, il va aller chercher une nouvelle photo du flux vidéo (qui sera forcément plus à jour que si nous stockons dans un tampon (« buffer ») certaines images)

Explication détaillée du thread de détection de la voix:

Le diagramme ci-dessous en illustre le principe :

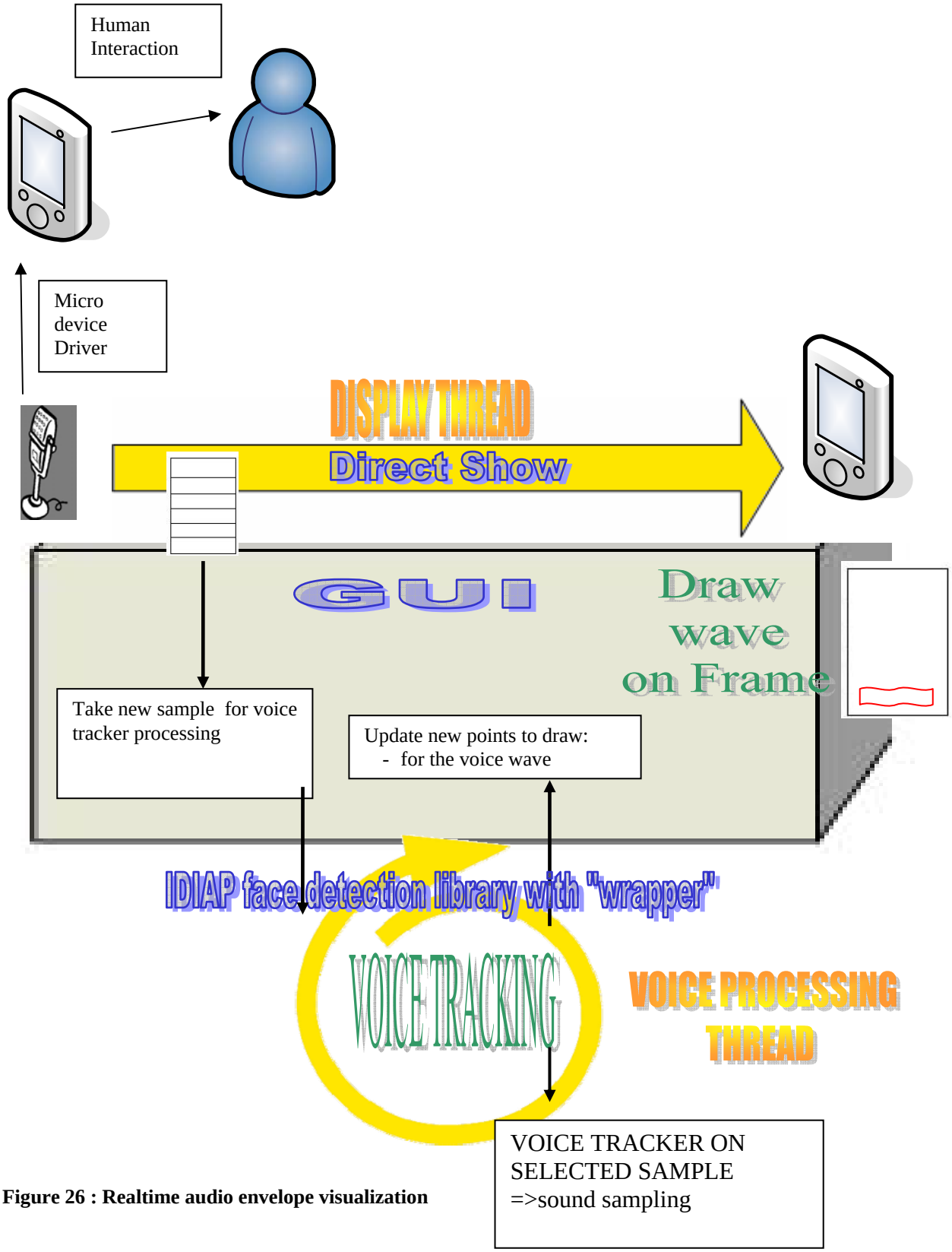


Figure 26 : Realtime audio envelope visualization

Explications

Comme indiqué plus haut, cette partie n'a pas été traitée dans le projet de diplôme, nous allons uniquement lancé des pistes de réflexion pour le « voice tracking »
L'explication est assez proche de celle du système de détection faciale. En effet, le principe reste le même, mis à part le fait que cette fois il s'agit du son.

Le « display thread » est le processus d'affichage ; il va se charger d'afficher les images les unes après les autres sur l'output du système qui est l'application avec le « stream vidéo » de la caméra. Il va également offrir la possibilité au GUI de lui « voler » un sample audio et de l'enregistrer comme un fichier « *.wav » sur le PDA.
Bien entendu, le code de DirectShow du « display thread » devra être adapté en conséquence afin de pouvoir séparer l'image du son.

Si le voice tracker de l'IDIAP est semblable au face tracker alors nous pouvons supposer ceci :
Le GUI va se charger de prendre un « sample » du flux audio pour la détection de voix en lançant la fonction d'enregistrement au format « *.wav » du « display thread »
Il va ensuite passer ce fichier « *.wav » au « voice tracking thread » qui va lui retourner un booléen ainsi que des entiers. Ce booléen indique si oui ou non la voix d'une personne a été trouvé et les entiers donnent les coordonnées de l'enveloppe vocale à dessiner.
Ce GUI va se charger de contrôler la valeur du booléen. Si la valeur du booléen est '1' alors la voix d'une personne a été détecté, sinon il vaudra '0' et aucune voix n'a été identifiée par le détecteur de voix.
Si une voix a été trouvée, le GUI va se charger de dessiner et mettre à jour sur le flux vidéo les coordonnées des points représentant l'enveloppe vocale.
Si aucune voix n'a été détectée, bien entendu, il ne sera affiché qu'une ligne droite.
Toutefois ce thread ne se charge pas de détecter la voix, Il ne se charge que de dessiner et rien de plus!
Le « thread de détection vocal » se charge de faire le traitement de « samples » audio à l'aide de l'algorithme de détection de la voix fourni par l'IDIAP.

Vision idéale :
Cette opération de traitement du son va se faire sur un "sample" parmi M. L'objectif serait d'avoir une fréquence d'échantillonnement du flux audio se trouvant environ à 8 [kHz] afin de faire d'être plus ou moins en accord avec la réalité entre la personne qui parle et le "voice tracking".

Vision réelle :
Ce cas de figure idéale ne pourra être respectés que si temps nécessaire au traitement de la voix et à l'acquisition des données ne dépassent pas 125 [µs].

La philosophie du voice tracking sur windows mobile reste actuellement en suspens et demande une plus grande analyse dans le cadre d'une implémentation future.

Structure du programme

Inputs Outputs des blocs

Nous allons ici voir la structure complète du programme en nous focalisant sur les *inputs* et *outputs* de chacun des blocs.

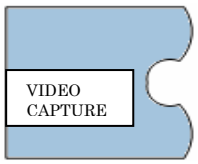
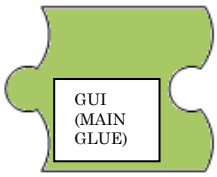
Blocs \ *Eléments*	***Inputs***	***Outputs***
	- connexion au driver de la camera du PDA	- rendre le flux vidéo sur une fenêtre de prévisualisation « preview window » - prendre une image de ce flux vidéo et l'enregistrer dans un fichier .bmp
	- les fonctions exportés par les DLL de capture vidéo et de face tracking - une image non 24 bits à convertir en 24 bits	- mise en place de l'application finale - dessin sur le flux vidéo - mettre sur pied une interface utilisateur
	- le fichier image 24 bits où se trouve le visage à détecter - un modèle de détection de visage ayant toutes les propriétés et définitions d'un modèle universel de visage. Ce fichier se nomme « frontal.model » dans la nouvelle librairie et « faceDetection.vmw » pour l'ancienne librairie	- le résultat de la détection de visage (un visage a-t-il été trouvé ou non ?) - si un visage a été trouvé, il s'agit de connaître les coordonnées (x,y) en pixels de la position du visage et des yeux - un score permettant de déterminer la fiabilité de la détection (dans le cas d'une future application d'authentification)

Figure 27 : Input et Output du système « Face Tracker »

Schéma fonctionnel du programme

Etape (1) :

L'utilisateur de l'application interagit avec celle-ci à l'aide d'une interface graphique (GUI).

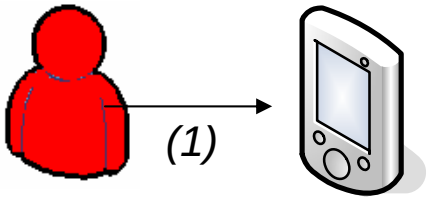


Figure 28 : Interaction utilisateur- programme

L'application va se lancer

Etape (2) et (3) :

Une fois l'application lancée :
Le bloc de capture vidéo (qui abstrait toutes les connexions faites par DirectShow) va se charger de faire l'acquisition du flux de donnée vidéo. Il va de plus diriger le flux vidéo vers deux directions :

- sur une fenêtre vidéo
- vers un fichier .bmp

La fenêtre vidéo va permettre d'avoir une prévisualisation de ce que voit la caméra. Et quand on lui en donne l'ordre, ce même bloc va prendre une image du flux vidéo pour l'enregistrer vers un fichier.

Voilà ce que fait le bloc vidéo capture :

- Il se charge de se connecter au driver de la caméra du PDA, et de transférer le flux vidéo provenant de cette caméra vers une fenêtre de preview *(2)*
- Lorsqu'on lui demande il prend une image de ce même flux pour l'enregistrer vers un fichier .bmp *(3)*

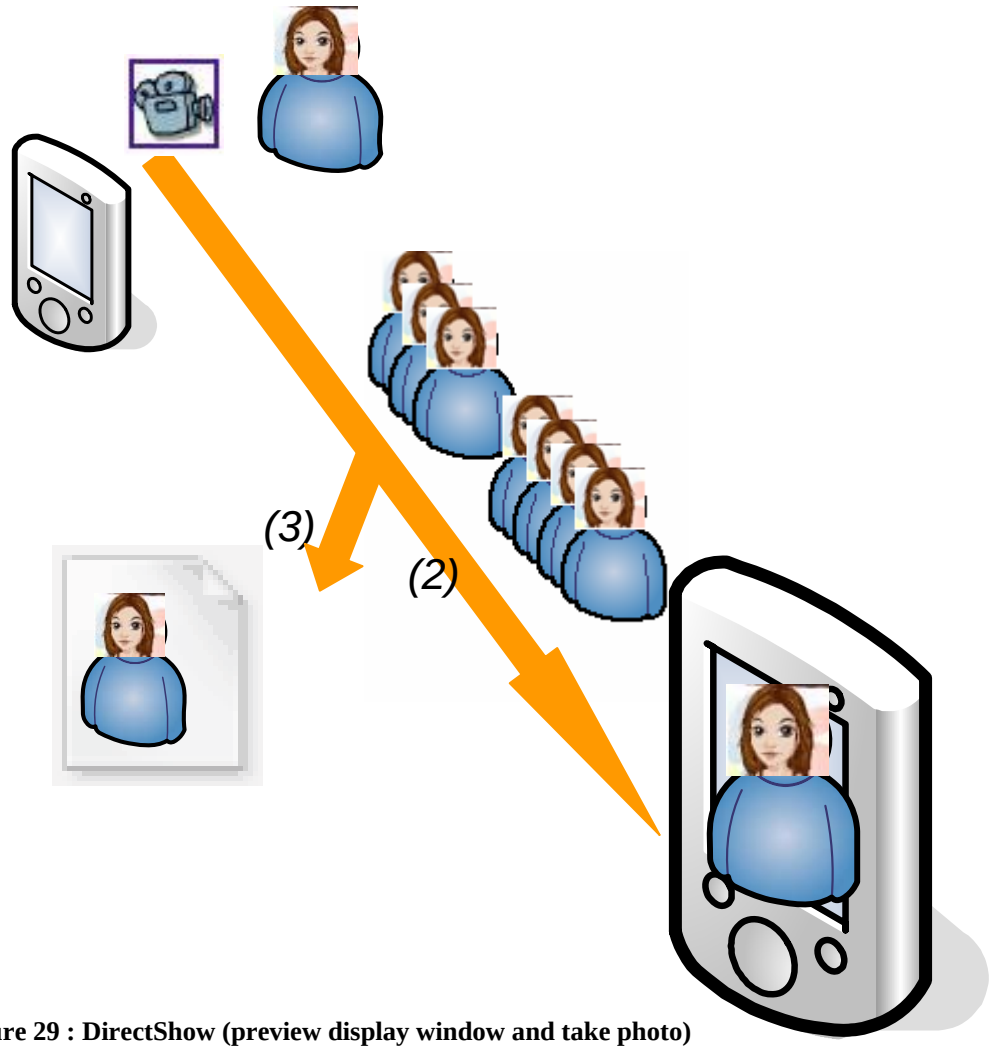


Figure 29 : DirectShow (preview display window and take photo)

Etape (4):

Le GUI va réenregistrer l'image BMP de l'étape *(3)* dans un fichier standard BMP RGB 24 bits.

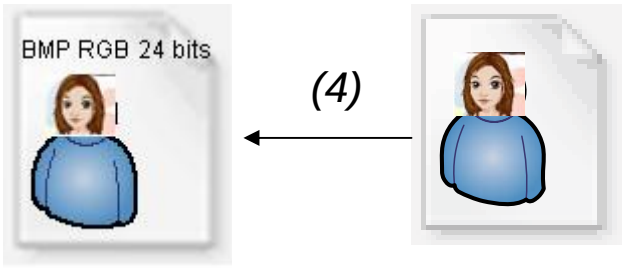


Figure 30 : Convertisseur d'image BMP en standard BMP RGB 24 bits

Etape (5):

Le bloc de détection de visage va prendre un fichier bitmap (il suppose qu'il a toujours un fichier bitmap bmp RGB 24) sur lequel il réalise une détection de visage en se basant sur un fichier modèle de détection de visage.

Si un visage a été détecté, il indique qu'un visage se trouve bien sur l'image retourne les coordonnées (x,y) en pixels de l'image où se trouve le visage et les yeux.

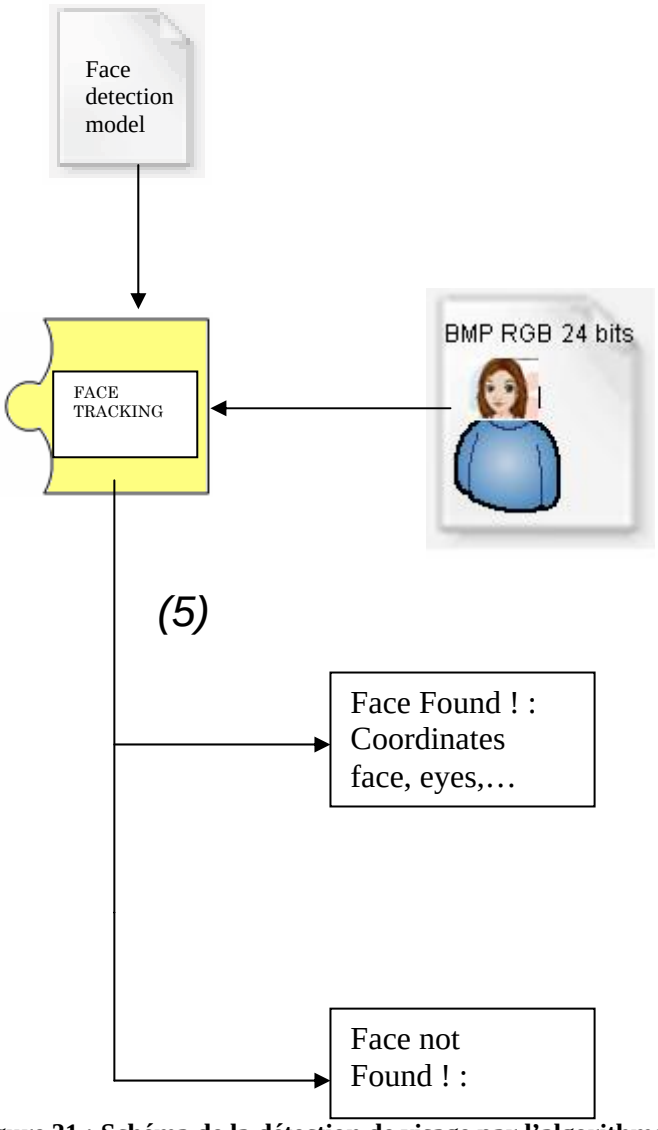


Figure 31 : Schéma de la détection de visage par l'algorithme de l'IDIAP

Etape (6):

Le GUI va faire appel à la fonction de détection de visage et donc connaître le résultat de la détection.

Si un visage a été trouvé, le GUI va intégrer et mettre à jour les nouvelles coordonnées du visage et des yeux dans une structure de type « visage ».

Si aucun visage n'a été trouvé, le GUI va mettre à jour les nouvelles coordonnées du visage et des yeux dans la structure avec des '0'.

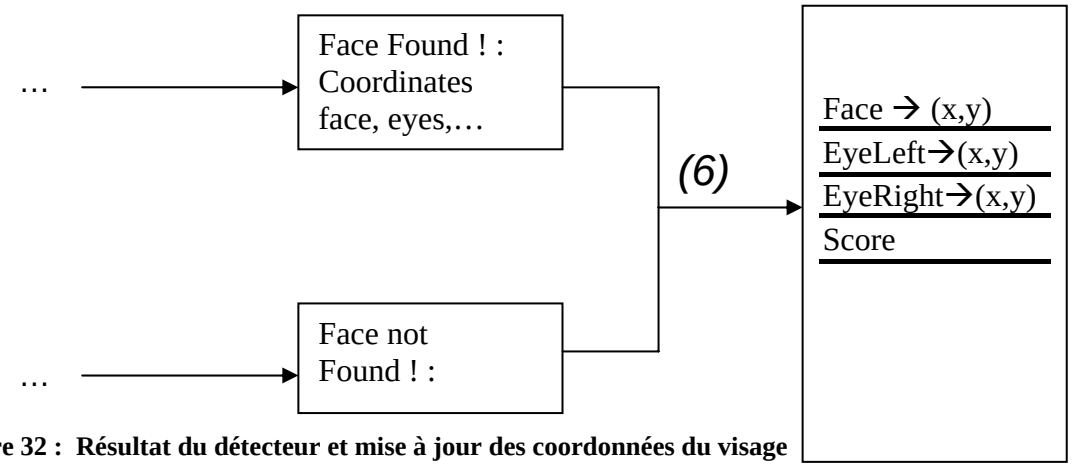


Figure 32 : Résultat du détecteur et mise à jour des coordonnées du visage

Etape (7):

Un visage à été détecté :

Le GUI va redessiner 3 carrés (représentant le visage et les yeux) sur le flux vidéo avec les coordonnées de l'image redimensionnée pour être adaptée à la fenêtre du flux vidéo

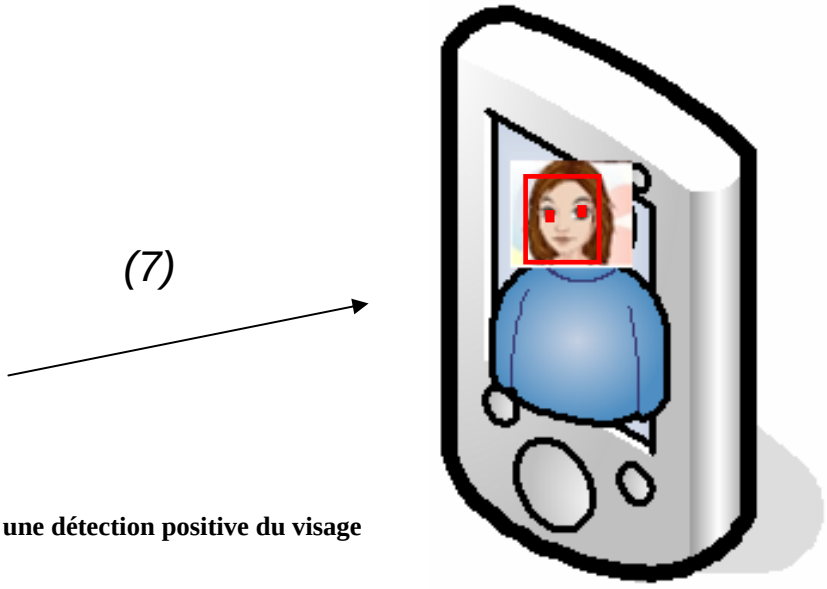


Figure 33 : Dessin des carrés après une détection positive du visage

Fonctionnalité du programme

Au démarrage du programme, nous tombons sur un écran d'invitation avec le logo « Face Tracker » développé pour cette application :

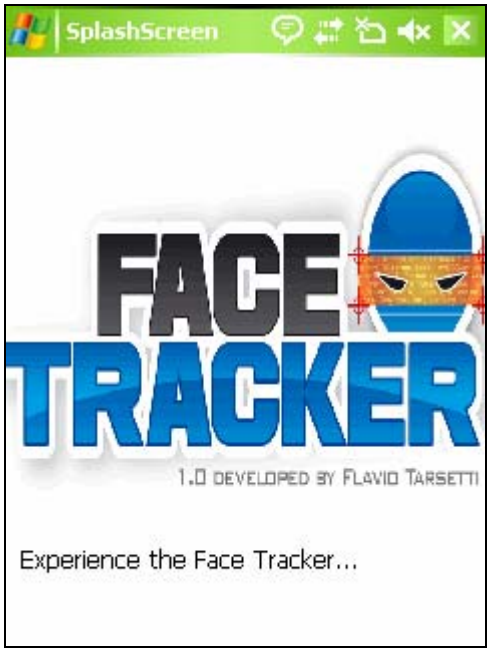


Figure 34 : SplashScreen image

Nous tombons ensuite sur l'écran principal suivant où nous avons le choix entre cocher ou pas la « `checkbox` » `verify`:

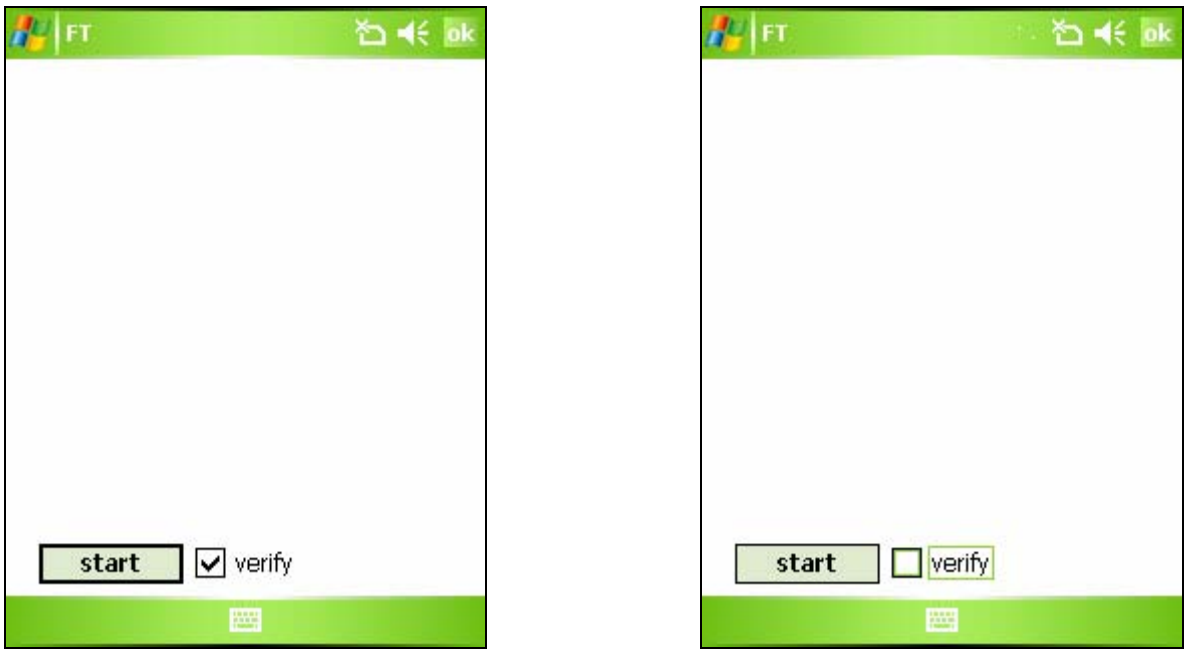


Figure 35 : Ecran principal de l'application

Si nous cochons la « `checkbox` » et que nous appuyons sur le bouton `play` :

Une fenêtre de prévisualisation des données en provenance de la caméra est affichée :

Figure 36 : appui sur le bouton *start*

Au moment ou un visage est détecté un carré va se dessiner sur le flux vidéo:
Cependant, étant donné que la mise à jour sur le flux vidéo prends environ entre 2 et 5 secondes en général, il se peut que nous pensions que les informations du visage détecté sont fausse. C'est dans cette optique qu'une petite fenêtre de prévisualisation a été implémentée afin de montrer l'image sur laquelle l'algorithme de détection de visage a été lancé. Cette petite fenêtre ne va à chaque fois montrer uniquement que le dernier visage qui a été détecté.

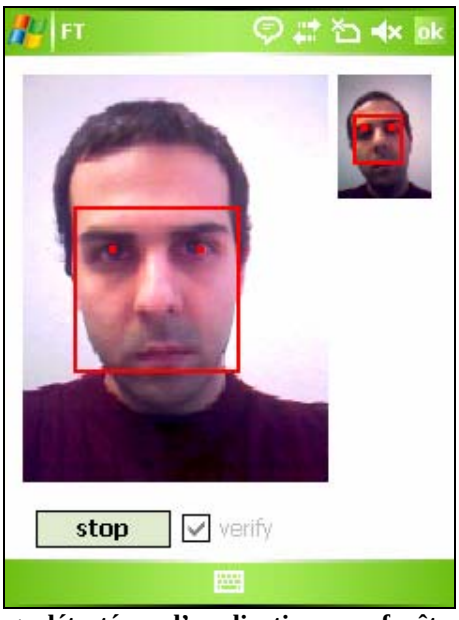


Figure 37 : Visage détecté par l'application avec fenêtre de vérification

Si nous appuyons sur le bouton stop le flux vidéo va s'arrêter mais la fenêtre de vérification reste ouverte et montre le résultat de la dernière détection réussie

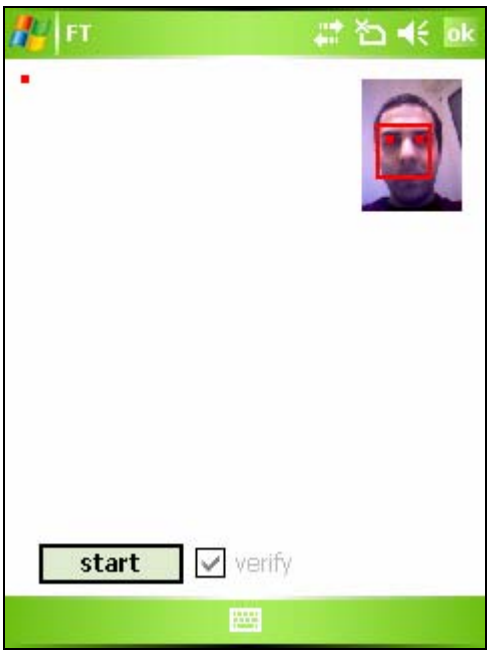


Figure 38 : appui sur le bouton stop après que l'application ait démarré et détecté un visage

Dans le cas où nous n'avons pas cliqué sur le bouton de la « checkbox », l'application va se charger de trouver la plus grande taille possible cohérente pour le « preview window » du flux de données vidéo :

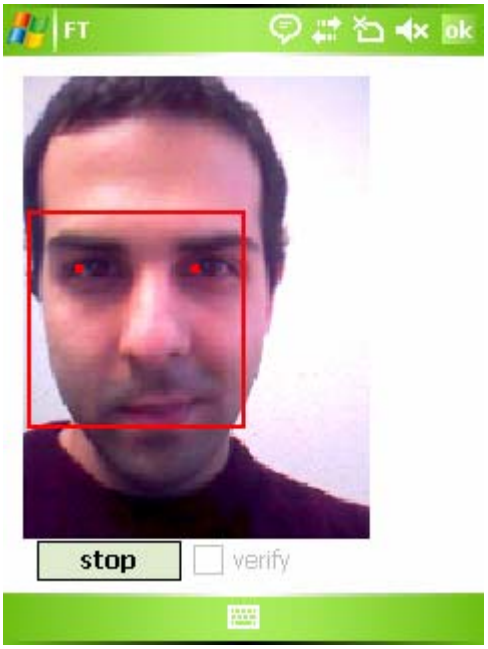


Figure 39 : détection de visage d'un flux vidéo sans la fenêtre de vérification

Le diagramme d'état transition (Statechart Diagramm) résume assez bien les cas d'utilisations auxquels l'utilisateur de l'application peut se trouver confonter :

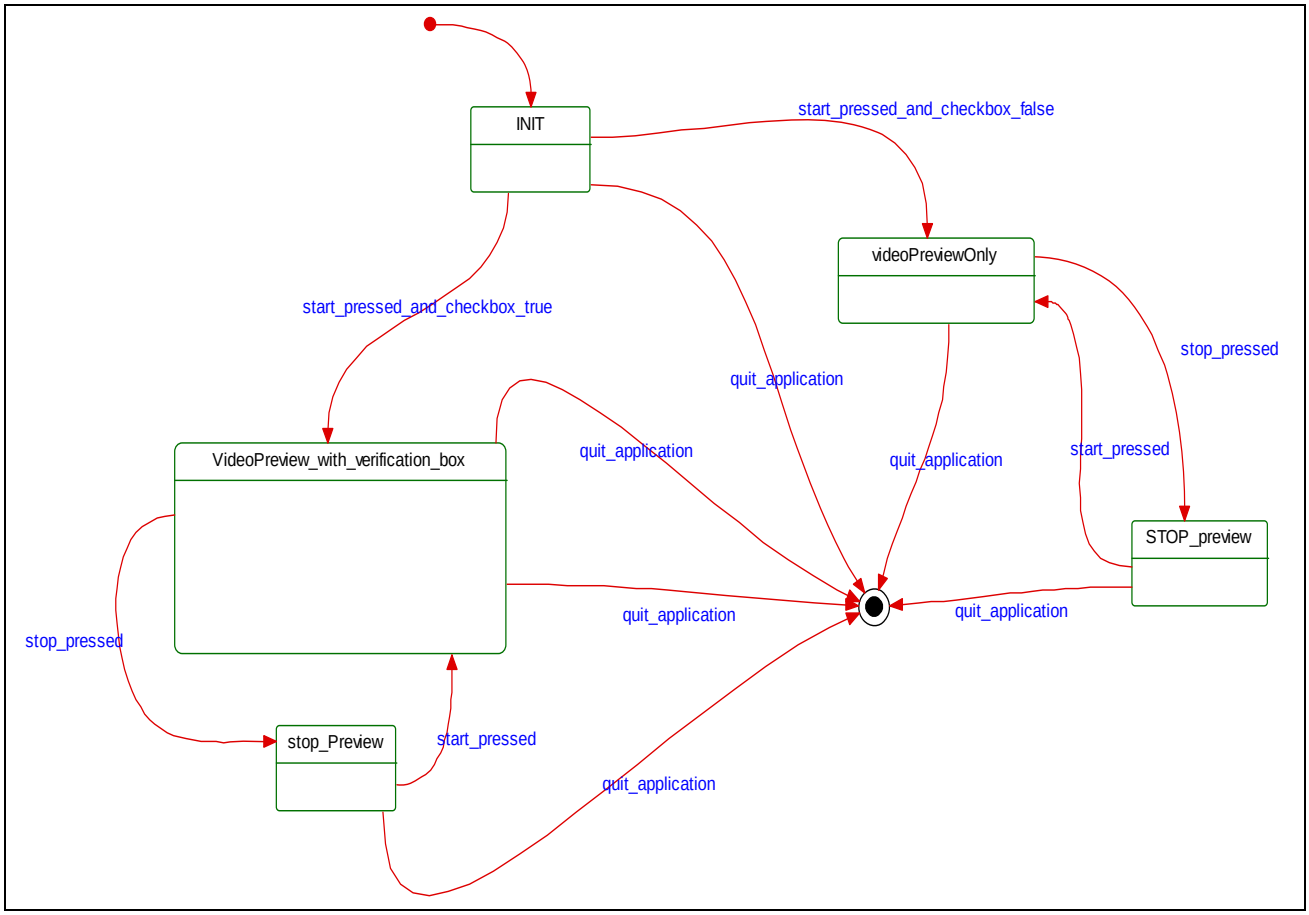


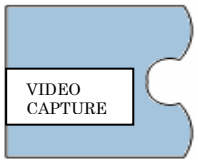
Figure 40 : Diagramme d'état transition des cas d'utilisation du programme

Bien entendu le diagramme d'état transition de la partie GUI gérable par l'utilisateur est plutôt limité pour l'instant, l'objectif étant plutôt de réaliser un démonstrateur dans le but de prouver si oui ou non une application de ce genre puisse fonctionner sur un téléphone mobile.

Architecture du programme

Nous allons dans cette partie reprendre la structure du projet et revoir de manière détaillée chacun des blocs du projet « Face Tracker sur Windows Mobile »
Nous allons traiter :
- de la capture vidéo
- du face tracking
- du GUI

Capture Vidéo



VIDEO CAPTURE

L'application va employer DirectShow pour réaliser la capture audio et vidéo. Nous remarquons la structure typique des applications "DrectShow", à savoir : l'emploi des multiples filtres dans un graphe de filtre.

DirectShow sur Windows Mobile

Direct Show pour un système Windows Mobile est très différent du DirectShow que l'on utilise sur des systèmes comme Windows XP (par exemple). Toutefois il comporte de nombreux points communs qu'il convient de ne pas négliger.
Un grand nombre de PDA possèdent depuis quelques temps une caméra. DirectShow est apparue afin de permettre aux développeurs de créer des applications pouvant interagir avec ces caméras embarquées. Depuis la version 5 de Windows Mobile, une intégration de l'API de DirectShow dans le SDK des systèmes d'exploitations Windows Mobile 5 et 6 a été implémentée. DirectShow fournit ainsi aux développeurs la possibilité d'accéder et d'avoir un contrôle sur le hardware de la caméra, permettant la capture de photo ou de streaming vidéo. DirectShow permet de faire l'abstraction des détails du hardware de chaque caméra fournissant ainsi aux développeurs d'application une simple, commune et consistante interface de programmation.

DirectShow supporte une grande variété de formats de fichiers, et les développeurs peuvent les adapter et les étendre afin des les faire soutenir des formats nouveaux ou spécialisés. Similaire à Direct3D et DirectDraw, DirectShow est une implémentation embarquée de l'API de la version « Desktop » des ordinateurs de bureau. Les développeurs de la version « Desktop » peuvent ainsi très rapidement appliquer leurs connaissances sur le monde embarqué moyennant un minimum d'investissement avec les petites différences dans l'implémentation du code.

DirectShow fait donc partie des supports avancés fournis par le SDK de Windows Mobile. Il fournit un grand contrôle et une grande flexibilité. Il s'agit d'une API native, qui est donc disponible à travers le toolset natif de Visual Studio.

DirectShow sur Windows Mobile « Face Tracker »

L'idée de son implémentation dans le projet du « Face Tracker » est de pouvoir ainsi accéder à la caméra du PDA. Le but est de pouvoir contrôler un flux vidéo permettant d'une part d'afficher une fenêtre de prévisualisation en streaming de la

capture de la caméra, et d'autres part de prélever des images de ce flux vidéo que l'on va stocker afin de pouvoir être traitées.
L'idée est de stocker une image à chaque fois afin de pouvoir faire le traitement de la photo, c'est-à-dire, lancer le détecteur de visage et les algorithmes fournis par l'IDIAP sur la photo. Ceux-ci vont ainsi nous permettre de savoir si le visage d'une personne est présent ou non. Si c'est le cas, les algorithmes nous retournerons les coordonnées du visage sur l'image.
Il est à noter que le code de détection de l'IDIAP (fournie à travers la librairie Torch) demande à avoir une image en dégradé de gris. Il ne faut donc pas oublier de faire une reconversion de l'image en noir/blanc (grayscale).
Dans la nouvelle libraire de l'IDIAP (FaceFindingAPI_0.2.2), le grayscaling est fait par la libraire elle-même, il suffit uniquement de lui passer une image et de spécifier son format.

La partie acquisition d'image et contact avec le hardware de la caméra se fera en natif (unmanaged code), soit donc en C++. La partie de contrôle de l'application qui va lancer les threads importants de l'application comme celui s'occupant de l'affichage du flux vidéo (prévisualisation ou streaming vidéo) et capture d'une frame de ce flux se fera depuis du code managé, soit donc en C#. Ceci va permettre ainsi de bien distinguer les parties de l'application et d'être plus efficace dans le contrôle global de l'application.

Pour accéder à DirectShow depuis du code managé (ou « managed ») il existe plusieurs solutions :

- une solution est d'empaqueter le code non managé (ou « unmanaged ») qui fait appels à ses librairies dans une DLL native. Les fonctions de la DLL seront ensuite exportées dans le code managé.
- Une autre solution aurait été d'envelopper les composants COM de DirectShow dans une classe enveloppe (wrapping class) en utilisant l'intéropérabilité COM inclut dans le .NET Compact Framework 2.0. Il y a un projet open source qui se nomme « DirectShowNET Library » (que l'on trouve à l'adresse « http://directshownet.sourceforge.net/ ») qui fournit une classe enveloppe managé pour DirectShow pour la version ordinateur de bureau.

Le problème est qu'il n'existe actuellement aucune version disponible du projet « DirectShowNET Library » pour le .NET Compact Framework et ciblant du Windows Mobile.
Nous devons donc choisir la première solution, à savoir empaqueter les appels à DirectShow dans une DLL.

Explication d'une DLL

Une « bibliothèque logicielle » permet d'englober un certains nombres de fonctions qui peuvent être employés dans divers projets. Ces bibliothèques se différencient des exécutables par le fait qu'elles ne peuvent être exécutées directement. L'intérêt d'avoir des bibliothèques découle de la volonté d'employer directement du code que l'on ne souhaite pas devoir retaper à chaque fois.
Le .NET Compact Framework que nous utilisons dans le projet « Face Tracker » peut représenter dans ce sens une bibliothèque dans un sens plus large.
Une DLL est ce qu'on appelle une bibliothèque dynamique (dérivé de Dynamic Link Library). Celle-ci n'est pas incorporée dans l'exécutable final, mais ses fonctions peuvent être appelées depuis un autre programme.
Dans ce sens, une DLL peut être intéressante dans la mesure où elle sépare les parties bien distinctes d'un projet. Elle permet d'éviter une réécriture dans la mesure où la DLL peut être réutilisée dans un autre projet.

Si nous souhaitons développer du code utilisant DirectShow, il est important d'inclure 3 librairies dans les propriétés du projet. Il s'agit des librairies :

```
-dmoguids.lib
-strmiids.lib
-strmbase.lib
```

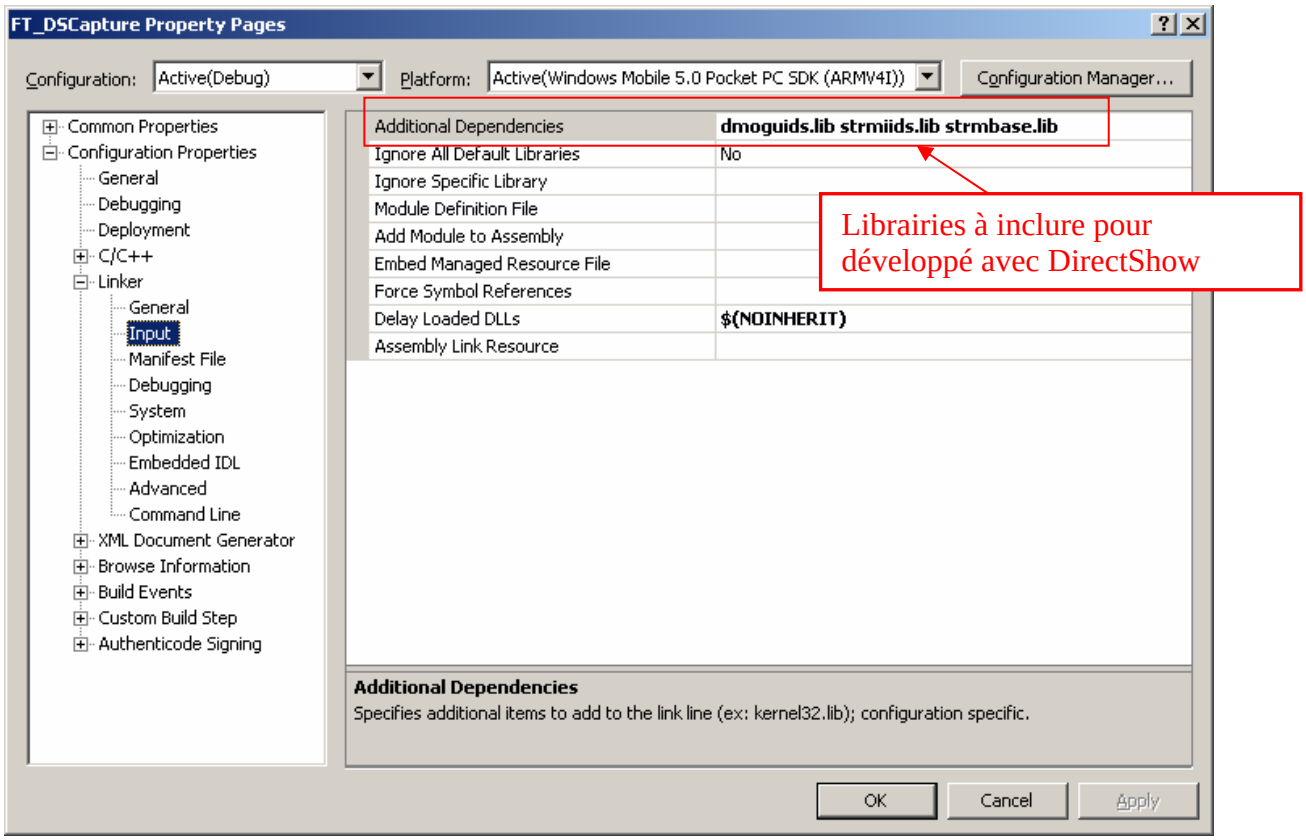


Figure 41 : Fichiers .lib obligatoire dans les projets DirectShow

DirectShow dans le projet de Face Tracking

La « Solution » du projet se présente ainsi :

3 projets dont le projet qui nous intéresse FT_DSCapture

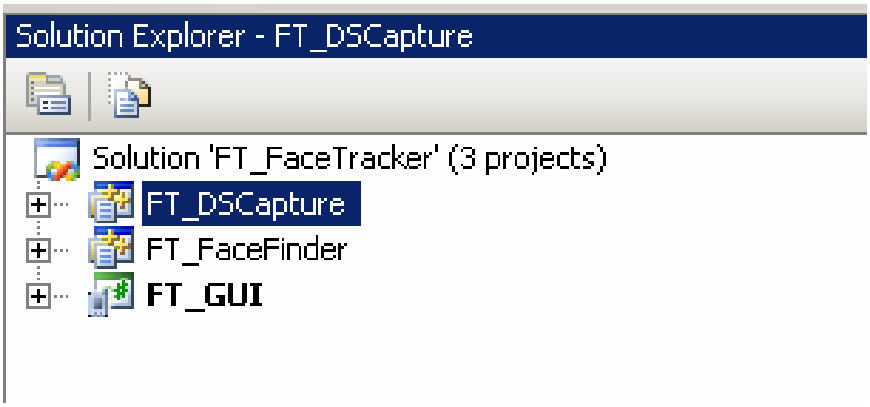


Figure 42 : Solution Explorer

Ouvrons la solution et plus particulièrement le projet FT_DSCapture :

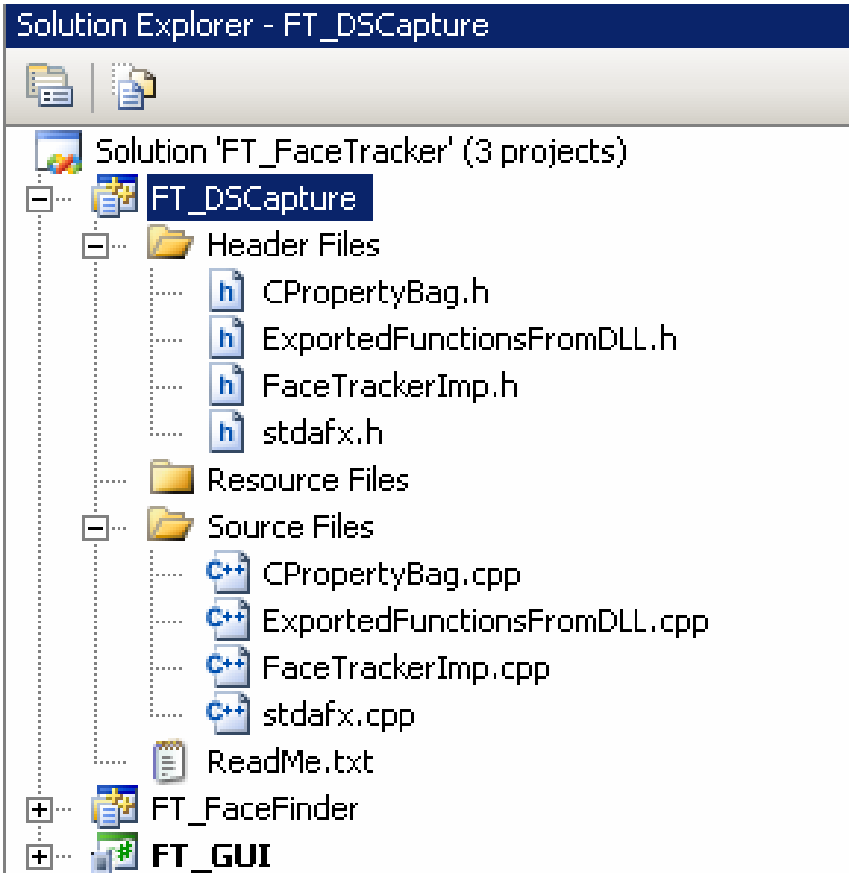


Figure 43 : Fichiers implémentés dans le projet FT_DSCapture

Nous remarquons un certains nombres de fichiers avec des fichiers d'en-têtes (« Headers Files ») et des fichiers sources (« Source Files »).

Nous pouvons déjà faire une séparation entre les fichiers qui touchent directement le monde DirectShow et ceux qui ne sont là que pour permettre une exportation simple des fonctions à utiliser.

Ainsi les fichiers regroupant l'implémentation de DirectShow sont les suivants :

« Header Files » :

- FaceTrackerImp.h
- CPropertyBag.h
- stdafx.h

« Source Files » :

- FaceTrackerImp.cpp
- CPropertyBag.cpp
- stdafx.cpp

Et les fichiers permettant une exportation aisée des fonctions depuis cette DLL sont les suivants :

« Header Files » :

- ExportedFunctionsFromDLL.h

« Source Files » :

- ExportedFunctionsFromDLL.cpp

Comme expliqué dans l'introduction sur DirectShow, celui-ci se base sur la notion de « graphe de filtres » pour interconnecté tout les éléments amener aux fonctionnalités de capture vidéo désiré.

Pratiquement toutes les applications utilisant DirectShow (pour ne pas dire toutes les applications) utilisent des filtres.

Par « filtre », nous entendons, les objets de connexions qui peuvent en général être regroupé au travers de 3 grands groupes :

- les filtres sources
- les filtres de transformations
- les filtres de rendus

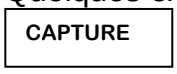
Les filtres sources permettent de « capturer » ou acquérir des données. (Exemple : capture vidéo)

Les filtres de transformations permettent, comme leur nom l'indique, de transformer des données. (Exemple : changement de couleur dans une image)

Les filtres de rendus permettent quant à eux de faire un rendu des données (Exemple : enregistrement des données dans un fichier)

Dans la page qui suit nous pouvons apercevoir la création des filtres et les connexions de ceux-ci dans le projet.

Remarque :
Quelques explications afin de mieux comprendre le schéma qui suit



: représente un filtre. Dans ce cas il s'agit d'un filtre de capture


: représente une pin de connexion entre deux filtres. A la sortie d'un filtre elle symbolise des méthodes qu'elle réclame (required). A l'entrée d'un filtre elle symbolise les méthodes fournies (provided)

PROGRAM STRUCTURE

Direct Show

CAPTURE GRAPH

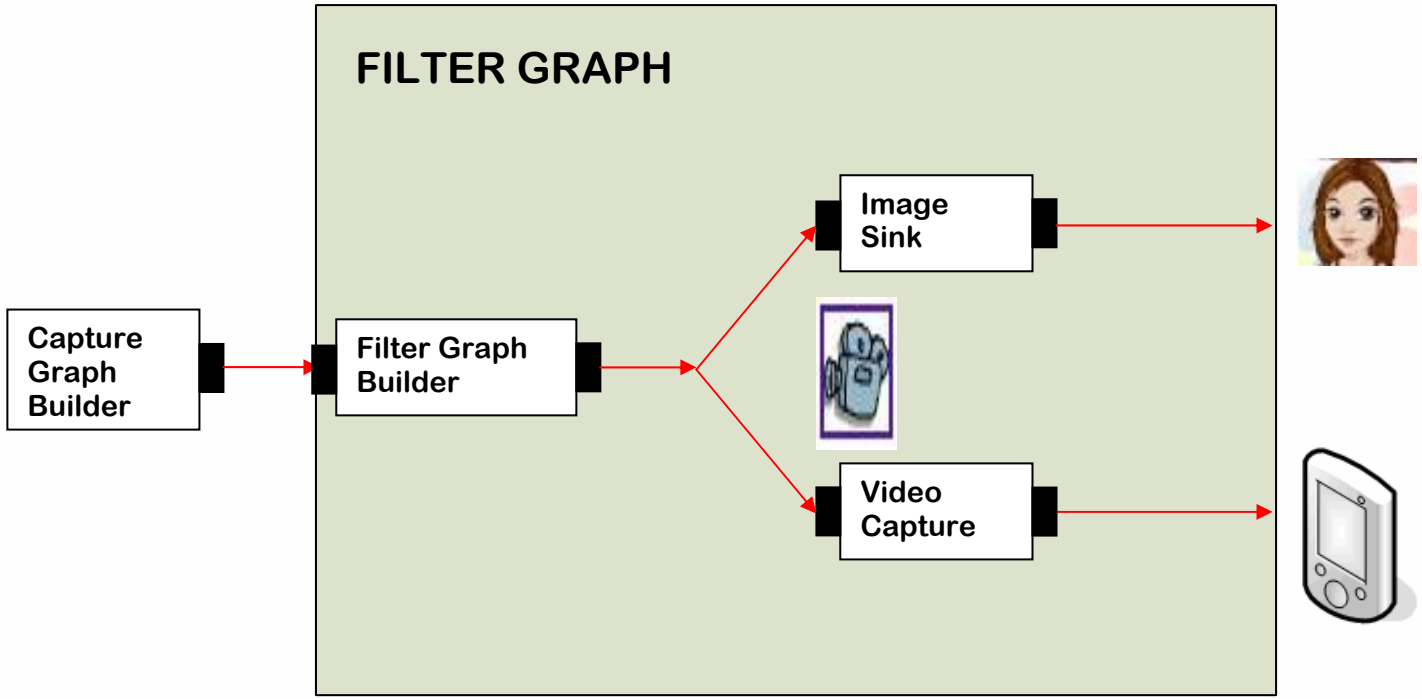


Figure 44 : DirectShow CaptureGraph -FilterGraph

Explications *:*

Le « Capture Graph Builder » est le filtre de base qui va permettre de créer un graphe qui se trouve être dans notre cas un graphe de capture.
Cependant le graphe de capture se trouve implémenter au travers d'un graphe de filtres.
Pour créer ce graphe de filtres, le « Capture Graph Builder » va donc avoir besoin d'un constructeur du graphe de filtres. C'est ainsi qu'il va pouvoir le créer au travers de l'instance « Filter Graph Builder ».

Le « Filter Graph Builder » est le véritable « architecte » des éléments relatifs à la capture vidéo de notre application.

Etant donné que nous souhaitons rendre les données provenant de la caméra de deux façons, à savoir :
- à l'aide d'une « preview window » des données de la camera
- à l'aide d'un fichier « *.bmp » pour prendre une trame du flux vidéo

Alors il va logiquement falloir implémenter deux filtres, soit donc :
- un filtre permettant de récupérer les données et les rendre directement (« Video Capture »)
- un filtre permettant de sauver des données dans un fichier « *.bmp ». (« Image Sink »)

Chemin des données dans le graphe

Maintenant que nous avons vu les connexions entre les divers filtres, nous pouvons voir à présent comment les éléments vont s'interconnecter entre eux au travers de leurs interfaces, et voir de façon plus spécifique le chemin des données dans le graphe.

Remarque *:*
Quelques explications afin de mieux comprendre le schéma qui suit

Les éléments sont représentés de manière équivalente au schéma précédant avec le rajout d'un nouvel élément :

: représente une interface spécifique au filtre. Dans ce cas il s'agit d'un contrôleur sur les données multimédias (vidéos dan notre cas).

: représente une connexion entre l'interface spécifique et le filtre

PROGRAM STRUCTURE

Direct Show

CAPTURE GRAPH

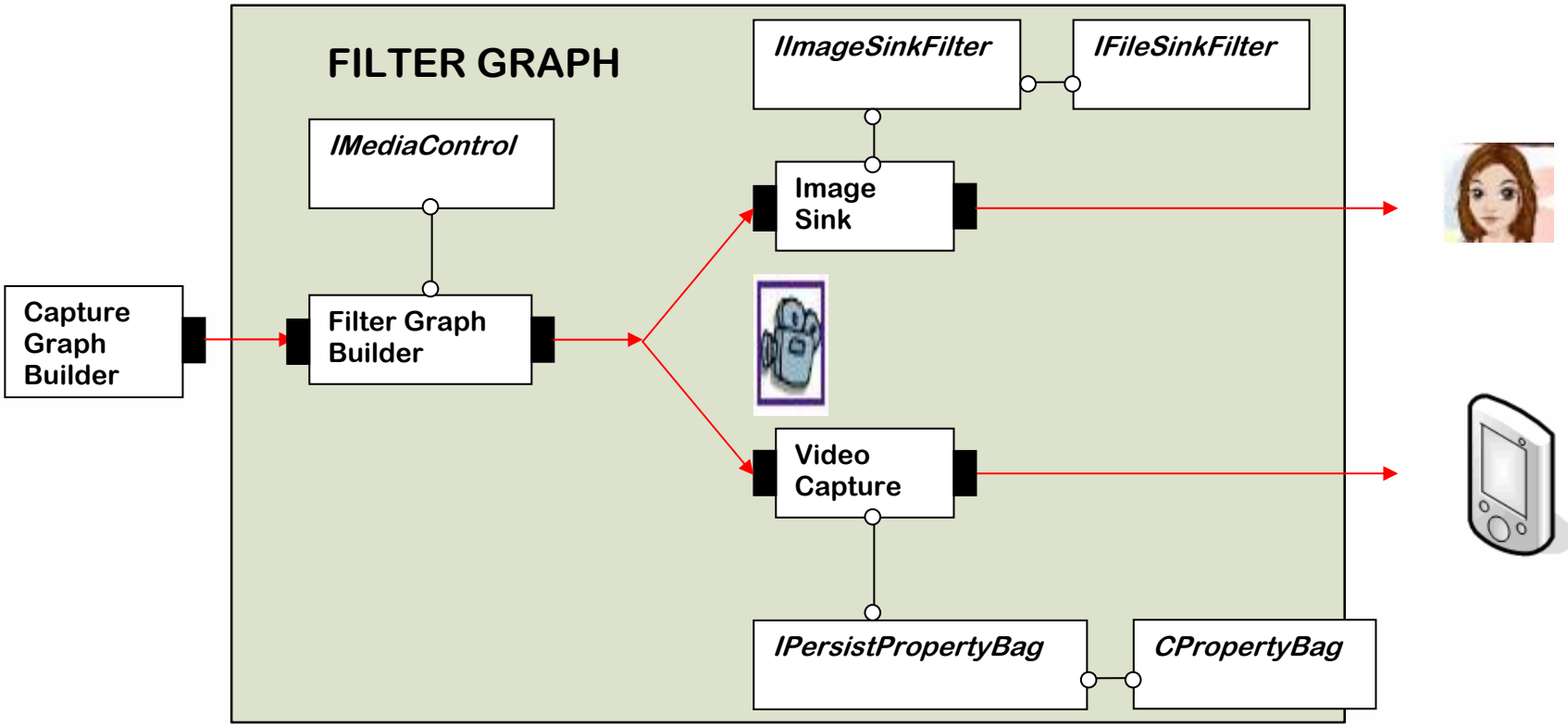


Figure 45 : DirectShow CaptureGraph - FilterGraph

Explications

Les filtres possèdent une ou plusieurs interfaces qui sont exposées et au travers desquels ils peuvent communiquer avec l'environnement qui les entoure.

IMediaControl

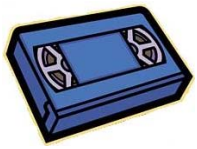
Le « Filter Graph Builder » va contrôler le flux de données grâce à son interface spécifique : IMediaControl

Le IMediaControl va permettre de donner des ordres au graphe en général.

Il faut percevoir le flux de donnée provenant de la caméra à la manière d'un magnétoscope et d'une cassette avec un film que nous souhaiterions visionner:

Le magnétoscope est système de visionnage du film :

Une fois la cassette à l'intérieur de l'appareil, le magnétoscope va pouvoir commencer à lire la bande magnétique.



Cependant l'utilisateur souhaite avoir le contrôle sur le système. Il possède ainsi une télécommande avec les boutons permettant de sélectionner le mode:

 : « ALLUMER – ETEINDRE » le système

 : « PLAY » pour démarrer le système ou lancer la lecture

 : « PAUSE » pour mettre le flux de donnée en pause

 : « STOP » pour arrêter les actions sur le flux de donner et le stopper

Ainsi le `IMediaControl` va jouer le rôle du contrôleur avec les opérations tel que : lancer la capture vidéo  , faire une pause  , stopper le flux 

IIMageSinkFilter - IFileSinkFilter

Le IIMageSinkFilter est l'interface qui expose des fonctions permettant d'accéder aux propriétés du filtre « Image Sink ». Il va se connecter à l'interface *IFileSinkFilter* qui, elle, fournit d'autres fonctions dont notamment celle de pouvoir écrire un flux de donnée multimédia dans un fichier.

Dans notre cas, cette interface va permettre d'enregistrer une trame du flux de donnée vidéo et de l'écrire dans un fichier « *.bmp ».

IPersistPropertyBag - CPropertyBag

L'interface IPersistPropertyBag expose des fonctions permettant de charger et sauvegarder les propriétés individuelles du filtre de capture vidéo « Video Capture ». Il va se connecter à l'interface CPropertyBag qui va fournir d'autres fonctions dont notamment celle permettant d'avoir accès au driver de la caméra du PDA.

Implémentation

Dans le code, nous allons retrouver les mêmes éléments que nous avons abordés dans les explications ci-dessus.

Filtres et Interfaces

Afin de regrouper tout ces filtres et interfaces, une structure de type `CameraFT` a été créé :

```
/*** DS Structure *************************************************************************/
struct CameraFT
{
        int                             bGraphRunning ; //specifies if the graph is running or not

        CComPtr<IGraphBuilder>          pFilterGraphBuilder ; //GraphBuilder object
        CComPtr<ICaptureGraphBuilder2>  pCaptureGraphBuilder; //GraphBuilder's object interface
        CComPtr <IBaseFilter>           pVideoCaptureFilter;  //Video Capture Filter

        //image capturing
        CComPtr<IBaseFilter>            pImageSinkFilter; //Image Sink Filter
        CComPtr<IImageSinkFilter>       pIImageSinkFilter;//Image Sink Filter's interface
        CComPtr<IFileSinkFilter>        pFileSink;        //File Sink Filter's interface

        //loading properties of an object
        //"IPersistPropertyBag provides an object with an IPropertyBag interface through which it
        //can save and load individual properties." (MSDN)
        CComPtr <IPersistPropertyBag>   pPropertyBag;
        CPropertyBag                    PropBag;    //needed to pass the capture device name
                                                    //to the filter

        //Data flow is controlled through the Filter graph's interface (IMediaControl)
        IMediaControl                   *pMediaControl ;


        CComPtr<IPin>                   pStillPin;
        CComPtr<IAMVideoControl>        pVideoControl;

} ;
```

Les filtres à implémenter sont les suivants :

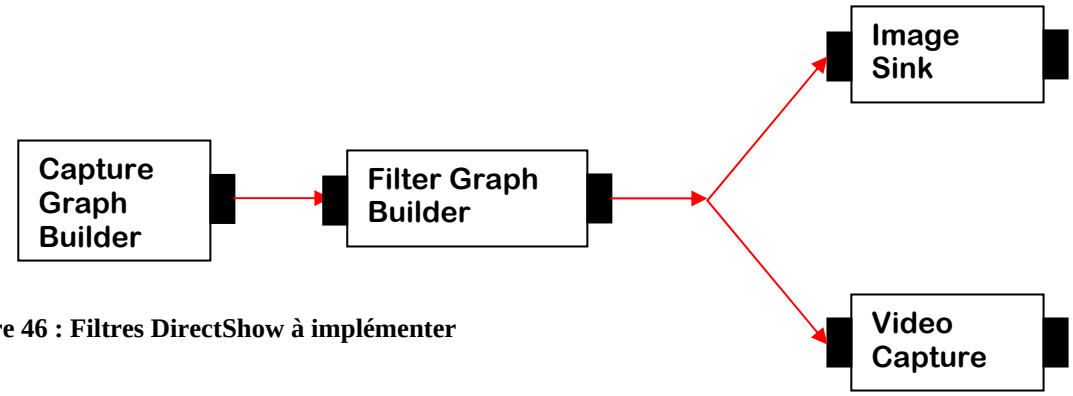


Figure 46 : Filtres DirectShow à implémenter

Nous allons donc tout d'abord les créer :
Création du graphe de capture (`pCaptureGraphBuilder`) et de son graphe de filtres (`pFilterGraphBuilder`) :

- `pFilterGraphBuilder` devient le graphe de filtres par défaut du graphe de capture `pCaptureGraphBuilder` à l'aide de sa méthode `SetFiltergraph()`

```
//Initialising all com objects*/
HRESULT hResult = S_OK ;

hResult = CoInitialize(NULL) ;
if(FAILED(hResult))
{
    MessageBox (NULL, _T("ERROR:COM Library not initialized!"),
                _T("ERROR"), MB_OK);
}

//Instance of the structure CameraFT
CameraFT *pCameraFT = new CameraFT() ;


//creating instances of IGraphBuilder and ICaptureGraphBuilder2
//using the COM function CoCreateInstance.
hResult = pCameraFT->pCaptureGraphBuilder.CoCreateInstance(CLSID_CaptureGraphBuilder) ;
if(FAILED(hResult))
{
    MessageBox (NULL, _T("ERROR:CaptureGraphBuilder not created!"),
                _T("ERROR"), MB_OK);
}
hResult = pCameraFT->pFilterGraphBuilder.CoCreateInstance( CLSID_FilterGraph );
if(FAILED(hResult))
{
    MessageBox (NULL, _T("ERROR:pFilterGraphBuilder not created!"),
                _T("ERROR"), MB_OK);
}

//setting the interface filter graph to be used by the capture graph builder
pCameraFT->pCaptureGraphBuilder->SetFiltergraph(pCameraFT->pFilterGraphBuilder) ;
```

Les filtres et interfaces sont des objets COM

Création du filtre de capture vidéo (`pVideoCaptureFilter`) :

```
/*Video Capture filters*/
hResult = pCameraFT->pVideoCaptureFilter.CoCreateInstance(CLSID_VideoCapture) ;
if(FAILED(hResult))
{
    MessageBox (NULL, _T("ERROR:VideoCaptureFilter not created!"),
                _T("ERROR"), MB_OK);
}
```

Ajout du filtre de capture vidéo (`pVideoCaptureFilter`) au graphe de filtre :

```
//adding VideoCaptureFilter to the Graph
hResult = pCameraFT->pFilterGraphBuilder->AddFilter( pCameraFT->pVideoCaptureFilter,
                                                     L"Video Capture source") ;
if(FAILED(hResult))
{
    MessageBox (NULL, _T("ERROR:pVideoCaptureFilter not added!"),
                _T("ERROR"), MB_OK);
}
```

Création du filtre de sauvegarde des données vidéo dans un fichier (`pImageSinkFilter`) et ajout de ce dernier au graphe de filtres :

```
//Image Sink Filter is used to write data from a Video Filter(video streaming)
//to a still image
hResult = pCameraFT->pImageSinkFilter.CoCreateInstance( CLSID_IMGSinkFilter );
if(FAILED(hResult))
{
    MessageBox (NULL, _T("ERROR:ImageSinkFilter not created!"),
                _T("ERROR"), MB_OK);
}

//adding Image Sink Filter to the Graph
hResult = pCameraFT->pFilterGraphBuilder->AddFilter( pCameraFT->pImageSinkFilter,
                                                     L"Still image filter" );
if(FAILED(hResult))
{
    MessageBox (NULL, _T("ERROR:pImageSinkFilter not added!"),
                _T("ERROR"), MB_OK);
}
```

Nous allons maintenant implémenter les interfaces de nos filtres :

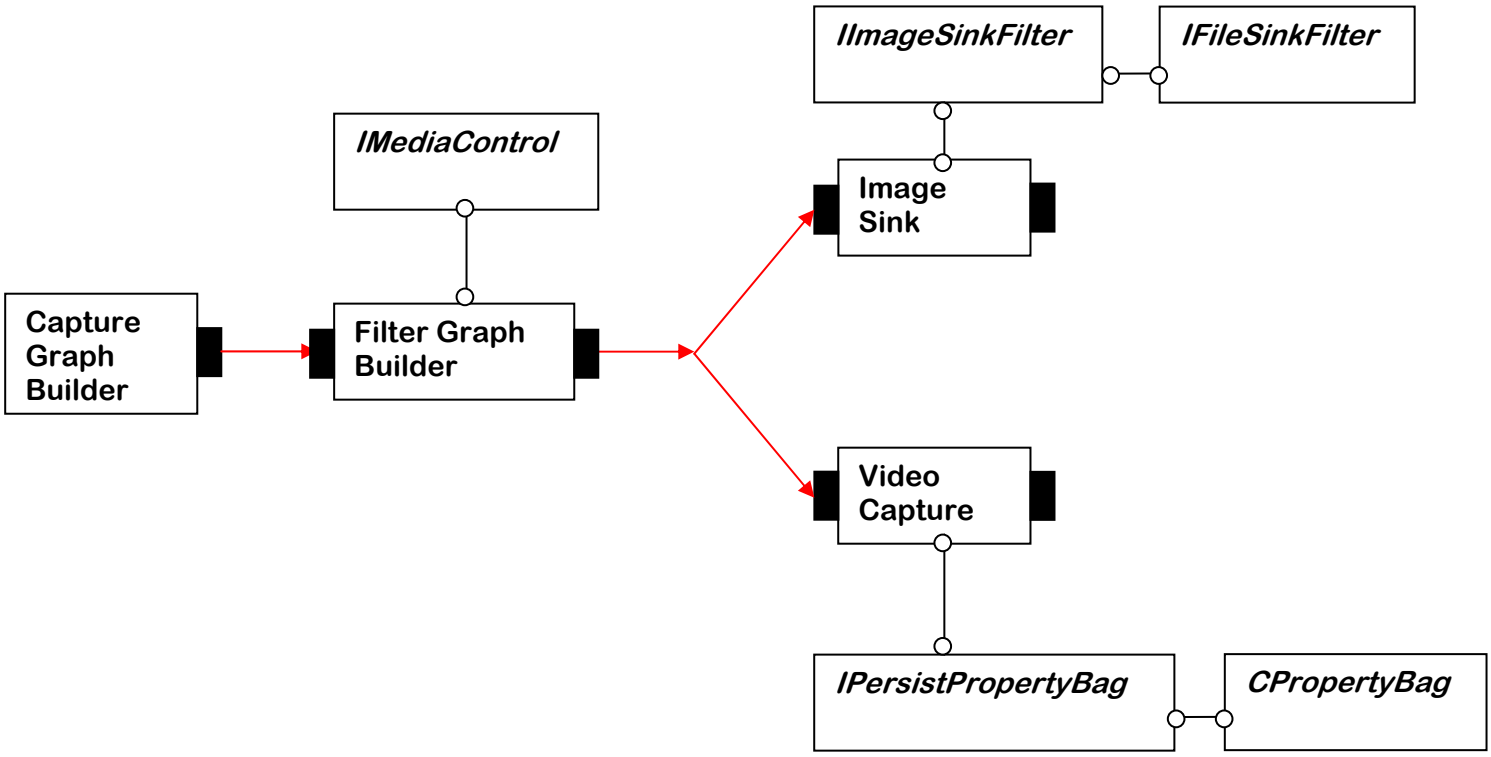


Figure 47 : Filtres et Interfaces DirectShow à implémenter

Ajout du contrôleur des données multimédia (`pMediaControl`) au graphe de filtres :

```
//Data flow is controlled through the Filter graph's interface (IMediaControl)
pCameraFT->pFilterGraphBuilder.QueryInterface(&pCameraFT->pMediaControl) ;
```

Ajout de l'interface `pIImageSinkFilter` exposant les fonctions permettant d'accéder aux propriétés du filtre « `pImageSinkFilter` » qui va se connecter à l'interface *`pFileSink`* exposant elle aussi des fonctions dont notamment celle de pouvoir écrire un flux de donnée multimédia dans un fichier :

```
//Image Sink Filter's exposed interface for setting it's properties
hResult = pCameraFT->pImageSinkFilter.QueryInterface(&pCameraFT->pIImageSinkFilter );
if(FAILED(hResult))
{
    MessageBox (NULL, _T("ERROR:pIImageSinkFilter interface not found!"),
                _T("ERROR"), MB_OK);
}

//File Sink Filter interface is added in order to write the media data stream
//into a file. The data here is a still image to be written in a file.
hResult = pCameraFT->pIImageSinkFilter.QueryInterface(&pCameraFT->pFileSink) ;
if(FAILED(hResult))
{
    MessageBox (NULL, _T("ERROR:pFileSink interface not found!"),
                _T("ERROR"), MB_OK);
}
```

L'interface `pPropertyBag` expose des fonctions permettant de charger et sauvegarder les propriétés individuelles du filtre de capture vidéo « `pVideoCaptureFilter` ». Il va se connecter à l'interface `PropBag` qui va fournir d'autres fonctions dont notamment celle permettant d'avoir accès au driver de la caméra du PDA.

```
//setting the capture device (here the capture device used is the camera ;) )
hResult = pCameraFT->pVideoCaptureFilter.QueryInterface( &pCameraFT->pPropertyBag ) ;
if(FAILED(hResult))
{
    MessageBox (NULL, _T("ERROR:VideoCapture queryInterface!"),
                _T("ERROR"), MB_OK);
}

//"VCapName" is an identifier for the Video Capture Filter device's property.
//firstCameraName() calls the function firstCameraDriver() in order to set a handle
//on the camera driver
pCameraFT->PropBag.Write(L"VCapName" , &firstCameraName()) ;

//Creating an association between Video Capture Filter and it's properties
pCameraFT->pPropertyBag->Load( &pCameraFT->PropBag , NULL) ;
```

Chemins des données

Jusqu'à présent, nous avons vu les connexions des filtres et leurs multiples interactions aux travers de leurs interfaces.

Une fois toutes les connexions réalisées, il faut tout de même encore veiller à créer le chemin qui permet de faire circuler l'information.

Pour rappel :
Les filtres sont regroupés en 3 grands groupes :

- les filtres sources
- les filtres de transformations
- les filtres de rendus

Les filtres sources permettent de « capturer » ou acquérir des données. (*Exemple* : capture vidéo)

Les filtres de transformations permettent, comme leur nom l'indique, de transformer des données. (*Exemple* : changement de couleur dans une image)

Les filtres de rendus permettent quant à eux de faire un rendu des données (*Exemple* : enregistrement des données dans un fichier)

Sachant que le flux de donnée circule à travers ces filtres, nous devons donc indiquer le chemin que le flux doit emprunter.

Nous souhaitons deux types de rendus :
- le rendu du flux de donnée sur une « preview window »
- le rendu d'une image provenant du flux de donnée

DirectShow est capable de rendre un flux de donnée en indiquant :

- le chemin que va prendre le « stream de donnée » depuis le filtre source en passant par le filtre de transformation jusqu'au filtre de rendu
- le type de donnée qui y transite
- la PIN de l'appareil qui doit être touchée

Par PIN, nous entendons la PIN exposée par le driver de la caméra du PDA.

Suivant l'appareil, le driver de la caméra va exposer 2 ou 3 PINS :

Cas du 2 PINS :
- une PIN pour la capture vidéo
- une PIN pour la capture d'une image fixe

Cas du 3 PINS :

- une PIN pour la capture vidéo
- une PIN pour la capture d'une image fixe
- une PIN pour le « preview » direct

Voici leurs schémas :

Schémas tirés du site http://msdn2.microsoft.com/en-us/library/ms940077.aspx en date du 20 novembre 2007 :

Schéma 2 PINS :

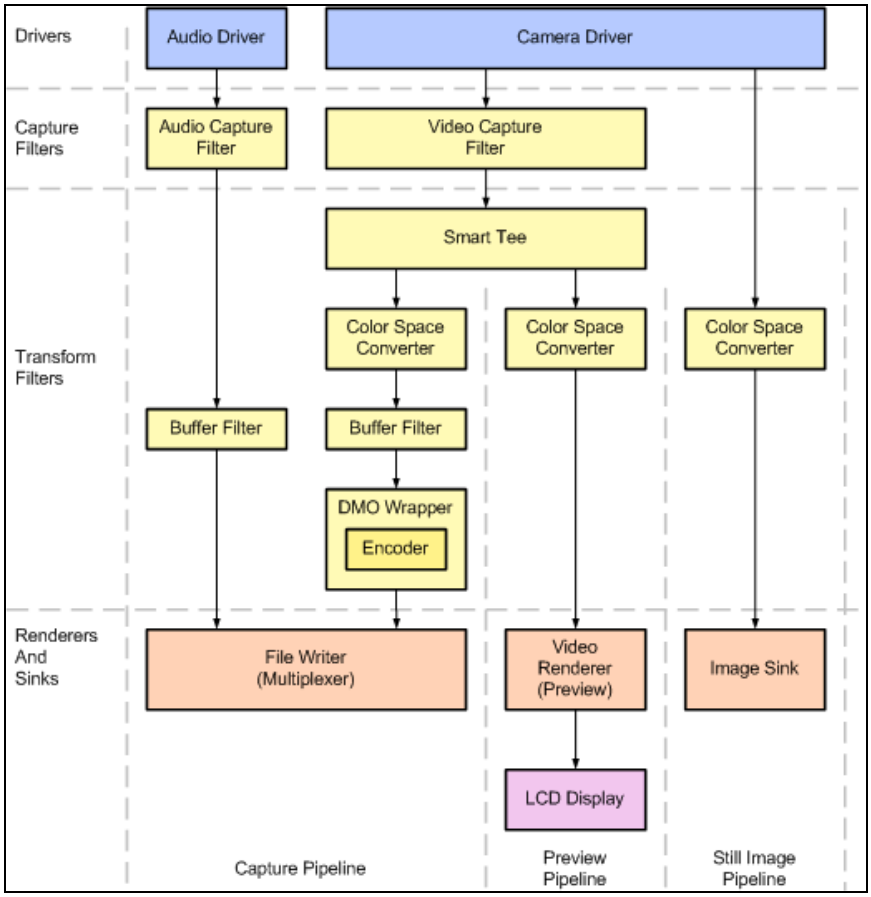


Figure 48 : schéma de connexions 2 pins exposées par le driver de la caméra

Schéma 3PINS

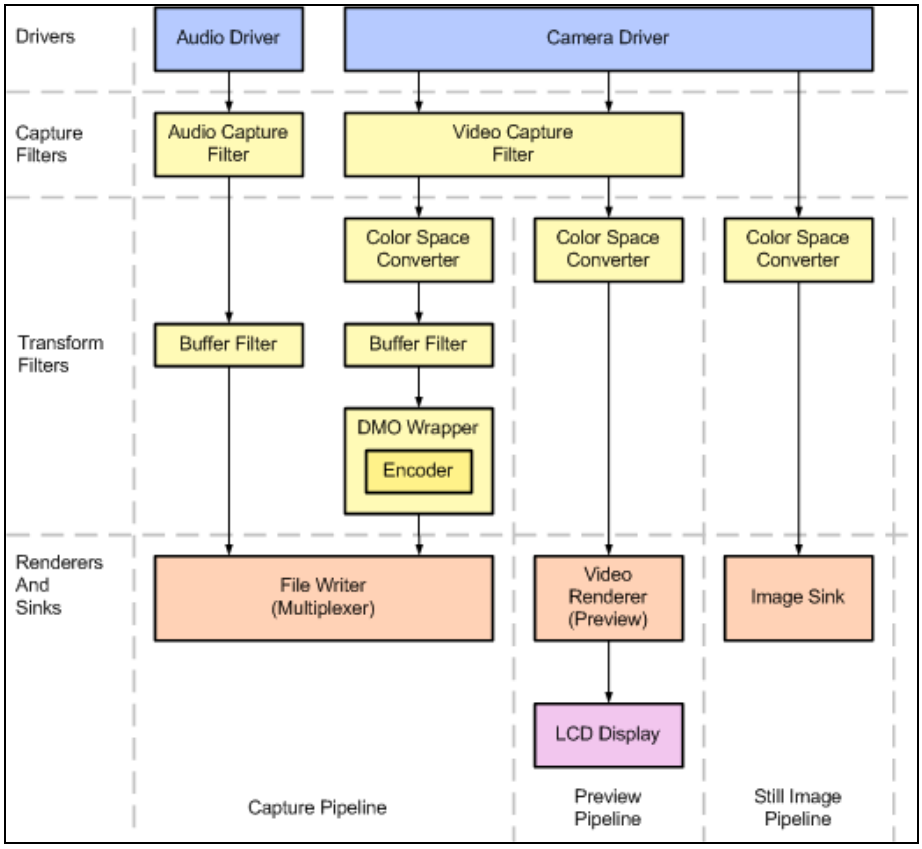


Figure 49 : schéma de connexions 3 pins exposées par le driver de la caméra

Cette distinction n'est véritablement essentielle que si nous souhaitons implémenter les 3 PINS alors que nous disposons que de 2.

Dans notre cas de figure nous n'allons pas tenir compte de cela et supposer que nous avons uniquement 2 PINS afin de pouvoir toucher un plus grand nombre d'appareil.

Reprenons donc notre explication sur le chemin des données au travers du graphique ci-dessous :

Remarque *:*
Quelques explications afin de mieux comprendre le schéma qui suit
Les nouveaux éléments suivants représentent :

 : Le chemin emprunté par les données

 : La pin qui a été touchée

Schéma du chemin des données

Figure 50 : Schémar de rendu normal des données(DirectShow)

Nous voyons clairement les 3 éléments essentiels, à savoir (PIN, filtres utilisés, data)

Schéma des deux chemins des données dans la partie Direct Show de l'application Face Tracker :

1er chemin :

Nous voulons avoir une « preview window »

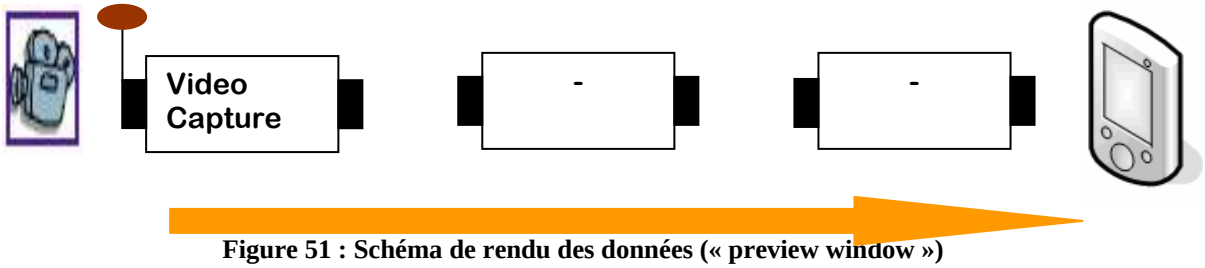


Figure 51 : Schéma de rendu des données (« preview window »)

Explications *:*

Nous voulons rendre directement ce qui sort de la caméra sur une fenêtre de « preview ».

Nous n'avons de ce fait ni besoin de transformer les données par un filtre intermédiaire ni de rendre de façon différente les données (car nous ne souhaitons pas rendre le flux de donnée dans un fichier de toute façon).

La PIN à utiliser pour le « preview » une `PIN_CATEGORY_PREVIEW`.

Le type de donnée multimédia transitant à travers les filtres est de type vidéo soit donc `MEDIATYPE_Video`

2ème chemin :

Nous voulons enregistrer dans un fichier une trame du flux vidéo

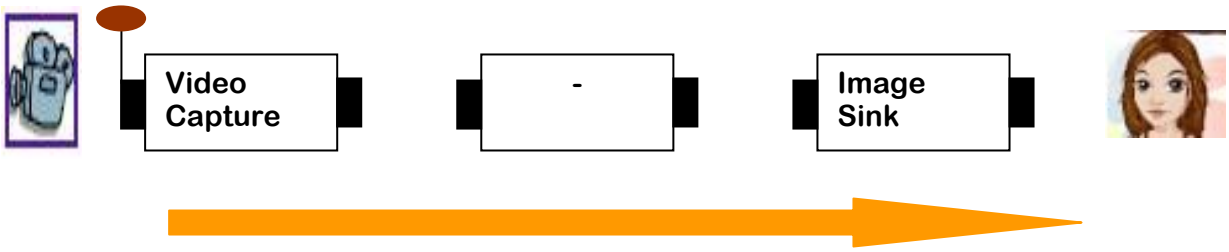


Figure 52 : Schéma de rendu des données (image stored in file)

Il est à remarquer que ce deuxième chemin est dépendant du premier dans la mesure où s'il n'y pas de capture vidéo, il n'y a pas de trame à prendre. C'est pourquoi il ne faudra lancer ce deuxième chemin de donnée uniquement dans le cas où tout s'est bien passé pour le premier chemin.

Etant donné que nous prenons des données de la capture vidéo, le filtre source sera bien évidemment le filtre « Video Capture ». Les données n'ont pas besoin d'être retransformées mais doivent être enregistrées dans un fichier, d'où l'ajout d'un filtre de rendu « Image Sink ».

La PIN à utiliser est celle permettant de capturer une image soit donc la `PIN_CATEGORY_STILL`.

Le type de donnée multimédia transitant à travers les filtres est toujours de type vidéo soit donc `MEDIATYPE_Video`.

Implémentation

Dans le code, nous allons retrouver les mêmes éléments que nous avons abordés dans les explications ci-dessus.

La fonction `RenderStream(PIN type, Media type, Source Filter, Transform Filter, Renderer Filter)` du constructeur de graphe permet de rendre les données.
Les paramètres de cette fonction sont les paramètres que nous avons spécifiés plus haut pour la réalisation d'un chemin de donnée.

1er chemin de donnée :

```
    //In order to accomplish a preview of the camera's data, we just need to render the data
    //captured from the VideoCaptureFilter.
    //Graph renders ----->   Source Filter    : Video Capture Filter,
    //                       Transform Filter : None
    //                       Renderer Filter  : None
    hResult = pCameraFT->pCaptureGraphBuilder->RenderStream( &PIN_CATEGORY_PREVIEW,
                                                             &MEDIATYPE_Video,
                                                             pCameraFT->pVideoCaptureFilter,
                                                             NULL,
                                                             NULL );


    if(FAILED(hResult))
    {
        //Video Stream is not rendered

        MessageBox (NULL, _T("ERROR:RenderStream"),
                    _T("ERROR"), MB_OK);

    }
    else
    {
        //Video Streams is rendered
```

Si le 1er cas c'est bien passé (soit donc dans le « `else` ») alors on peut créer le 2ème chemin de donnée :

```
        //In order to take a still image of the camera's data video stream, we need
        //to render the data from the VideoCaptureFilter to the ImageSinkFilter.
        //Graph renders ----->   Source Filter    : Video Capture Filter,
        //                       Transform Filter : None
        //                       Renderer Filter  : Image Sink Filter
        hResult = pCameraFT->pCaptureGraphBuilder->RenderStream( &PIN_CATEGORY_STILL,
                                                                 &MEDIATYPE_Video,
                                                                 pCameraFT->pVideoCaptureFilter,
                                                                 NULL,
                                                                 pCameraFT->pImageSinkFilter );
        if(FAILED(hResult))
        {
            MessageBox (NULL, _T("ERROR:pVideoCaptureFilter not added!"),
                        _T("ERROR"), MB_OK);
        }
```

Rendre le flux vidéo est une chose mais encore faut-il le rendre sur une fenêtre. Nous avons un flux vidéo, donc quoi de plus naturel que de le rendre à travers une « Video Window ».
La Video Window est accessible au travers de l'interface `gpVideoWindow` de type IID_IVideoWindow.

A cette fenêtre il va falloir donner les dimensions désirées.

Etant donnée qu'il s'agit d'une « fenêtre de preview » rectangulaire, cela peut se réaliser très simplement en déclarant un rectangle collée à la fenêtre cliente qui va accueillir l'image et donc spécifier automatiquement les dimensions de `gpVideoWindow`.

Le code ci-dessous explicite cette opération :

```
        //gpVideoWindeo is the interface to a Video Window
        //Our manager for this Filter Graph is calling this interface to set the properties
        //on a Video Window such as owner, etc...
        pCameraFT->pFilterGraphBuilder->QueryInterface(IID_IVideoWindow, (void **)&gpVideoWindow);

        //videoWindow is a child of the main window
        gpVideoWindow->put_Owner((OAHWND)pbowner) ;

        /*
        !!!Window Style should not be set -> application doesn't work properly !!!

        long myWindowStyle ;
        gpVideoWindow->get_WindowStyleEx(&myWindowStyle) ;
        gpVideoWindow->put_WindowStyleEx(myWindowStyle | WS_EX_TOPMOST) ;
        //gpVideoWindow->put_WindowStyleEx(WS_EX_TOPMOST) ;

        //gpVideoWindow->put_AutoShow(OATRUE) ;

        */


        //Rectangle Structure filled with coordinates of pbowner (the client area)
        RECT rect ;
        GetClientRect(pbowner, &rect) ;

        //LEFT - TOP - WIDTH - HEIGHT of the Video Window is specified
        //LEFT & TOP are set to zero because "coordinates are relative to a client area"(MSDN)
        gpVideoWindow->SetWindowPosition(0, 0, rect.right , rect.bottom) ;

        //window is visible & messages from pVideoWindow are posted in the main handler window
        hResult = gpVideoWindow->put_Visible(OATRUE) ;

        //if needed for mouse&keyboard messages
        //  gpVideoWindow->put_MessageDrain((OAHWND)pbowner) ;



        //graph is not running
        pCameraFT->bGraphRunning = 0 ;

        //setting the main handler to be used in this whole file
        *mainCameraHandler = (HANDLE)pCameraFT ;
```

Nous avons vu jusqu'ici la « philosophie » principale d'un programme DirectShow basé sur les filtres, interface et rendu de flux.

Toute cette partie de connexions et spécifications des rendus de flux ont été développées dans une fonction spécifique du « bloc » DirectShow à savoir, la fonction `cameraConnectAll`():

```
/*****************************************************************************************************/
/*** Function cameraConnectAll( HWND hwndOwner ,                                                   ***/
/***                           HANDLE *mainCameraHandler,                                          ***/
/***                           DWORD dwResolutionCamera,                                           ***/
/***                           HWND pbowner)                                                       ***/
/*****************************************************************************************************/
/*** This function is used to connect all the DirectShow's camera filters together                 ***/
/*** object for face tracking                                                                      ***/
/***                                                                                              ***/
/*** 4 parameters : 1) the main howner (referencing the window handler                             ***/
/***                2) the handler on the camera                                                  ***/
/***                3) the camera (photo) resolution                                              ***/
/***                4) the handler on the pictureBox(in the C# GUI)                               ***/
/*** return value : HRESULT( specifies if the connection went fine or not)                         ***/
/*****************************************************************************************************/

HRESULT faceTrackerImp::cameraConnectAll( HWND hwndOwner ,
                                          HANDLE *mainCameraHandler,
                                          DWORD dwResolutionCamera,
                                          HWND pbowner)
{
```

Les commentaires du code (en annexe) sont suffisamment pertinents pour comprendre les paramètres utilisés par cette fonction.

Nous allons voir maintenant les autres fonctions qui ont été développés dans le cadre de ce bloc sur DirectShow afin de satisfaire la fonctionnalité globale du bloc :

La fonction `updatePreview`() permet de créer si besoin est une autre `VideoWindow`

Cette fonction est utilisée par le GUI pour affichée une petite image permettant de voir la dernière reconnaissance de visage effectuée.

```
/*****************************************************************************************************/
/*** Function updatePreview(HWND pbowner)                                                          ***/
/*****************************************************************************************************/
/*** This function is used to launch another preview Window (used for the GUI small preview  )     ***/
/*** (used by GUI to show the "Face Tracking" on a still image)                                    ***/
/***                                                                                              ***/
/*** 1 parameter  : 1) the handler on the pictureBox(in the C# GUI)                                ***/
/*** return value : -                                                                              ***/
/*****************************************************************************************************/
VOID faceTrackerImp::updatePreview(HWND pbowner)
{
        //Rectangle Structure filled with coordinates of pbowner (the client area)
        RECT rect ;
        GetClientRect(pbowner, &rect) ;

                //LEFT - TOP - WIDTH - HEIGHT of the Video Window is specified
        //LEFT & TOP are set to zero because "coordinates are relative to a client area"(MSDN)
        gpVideoWindow->SetWindowPosition(0, 0, rect.right , rect.bottom) ;


}
```

Les fonction `firstCameraName()` et `firstCameraDriver()` ci-dessous, se charge de détecter une caméra. Et lorsqu'une caméra a été trouvée, on spécifie que l'on veut avoir une emprise sur la caméra à l'aide d'un « `handle` »

```
/*****************************************************************************************************/
/*** Function firstCameraName()                                                                    ***/
/*****************************************************************************************************/
/*** This function helps to put up an identifier of the Video Capture Filter device's property ***/
/*** (it calls the function firstCameraDriver() in order to set a handle on the camera driver) ***/
/***                                                                                             ***/
/*** 0 parameter  : -                                                                            ***/
/*** return value : CComVariant (represents the camera name)                                     ***/
/*****************************************************************************************************/
CComVariant faceTrackerImp::firstCameraName()
{
    WCHAR camDeviceName[ MAX_PATH + 1 ] ;
    CComVariant camName ;

    firstCameraDriver(camDeviceName) ;
    camName = camDeviceName ;

    return camName ;

}
```

```
/*****************************************************************************************************/
/*** Function firstCameraDriver(WCHAR *pName)                                                      ***/
/*****************************************************************************************************/
/*** This function finds the first camera driver available  & associates with it's properties  ***/
/*** The camera type is found by it's GUID exposed in file "camera.h" of WM5 -WM6 SDK           ***/
/*** Important Function used to set a handle on a camera driver.                                 ***/
/*** (this function is called by firstCameraName() function)                                     ***/
/***                                                                                             ***/
/*** 1 parameter  : 1) the camera name                                                           ***/
/*** return value : HRESULT (Finding camera status)                                              ***/
/*****************************************************************************************************/
HRESULT faceTrackerImp::firstCameraDriver(WCHAR *pName)
{
    HRESULT hResult = S_OK ;

    //getting a handle on the camera
    //DEVMGR_DEVICE_INFORMATION is a structure that helps us to get information on a device driver
    DEVMGR_DEVICE_INFORMATION       devInfo ;

    //Camera type is identified by its GUID
    //This GUID list can be found in the file camera.h of Windows Mobile 5 - 6 SDK
    GUID guidCamera = { 0xCB998A05, 0x122C, 0x4166, 0x84, 0x6A, 0x93, 0x3E, 0x4D,
                        0x7E, 0x3C, 0x86};

    devInfo.dwSize = sizeof(devInfo) ;

    //Finding the first suitable camera
    //search type is made by camera's GUID
    HANDLE handle = FindFirstDevice( DeviceSearchByGuid, &guidCamera, &devInfo ) ;
            if(( handle == NULL ) || ( devInfo.hDevice == NULL ))
    {
            MessageBox (NULL, _T("ERROR:nocam!"),
                    _T("ERROR"), MB_OK);
    }

    StringCchCopy(pName, MAX_PATH , devInfo.szLegacyName) ;
    //FindNextDevice( handle, &devInfo ) ;
    FindClose(handle) ;

    return hResult ; //Finding camera has encountered a problem?
}
```

Une fonction `setResolutionCamera`() a été implémentée et spécifie la résolution de l'appareil à utilisé dans le cas où l'appareil en expose plusieurs. Cette fonction n'a pas été utilisée dans le cadre du projet, mais pourrait aisément être utilisée (moyennant quelques modifications dans l'implémentation de son code)

Les fonctions ci-dessous sont utilisées par le contrôleur du flux vidéo : l'interface IMediaControl.

Une fonction `launchCapture`() permet d'allumer et de lancer le système :

 : « ALLUMER – ETEINDRE » le système

 : « PLAY » pour démarrer le système ou lancer la lecture

Une fonction pause() permet de mettre sur pause le « streaming» vidéo :

 : « PAUSE » pour mettre le flux de donnée en pause

```
/*****************************************************************************************************/
/*** Function pause(HANDLE handleCamera)                                                          ***/
/*****************************************************************************************************/
/*** This function pauses the preview by setting the MediaControl with "PAUSE" mode               ***/
/*** (!  be aware that the connections should be put up first!                                     ***/
/***     with the cameraConnectAll(...) function )                                                 ***/
/***                                                                                               ***/
/*** 1 parameter  : 1) the handler on the camera)                                                  ***/
/*** return value : HRESULT (specifies if pausing the capture went fine or not)                    ***/
/*****************************************************************************************************/
HRESULT faceTrackerImp::pause(HANDLE handleCamera)
{
    HRESULT hResult = S_OK ;

    CameraFT *pCameraFT = (CameraFT *) handleCamera ;

    hResult = pCameraFT->pMediaControl->Pause() ;

    return hResult ;
}
```

La fonction play() permet de redémarrer le système dans le cas où la fonction pause() a été appelé préalablement par exemple :

 : « PLAY » pour démarrer le système ou lancer la lecture

```
/*****************************************************************************************************/
/*** Function play(HANDLE handleCamera)                                                            ***/
/*****************************************************************************************************/
/*** This function plays the preview by setting the MediaControl with "RUN" mode                   ***/
/*** (to be used after a "PAUSE" mode for example)                                                 ***/
/***                                                                                              ***/
/*** 1 parameter  : 1) the handler on the camera)                                                  ***/
/*** return value : HRESULT (specifies if playing the capture went fine or not)                    ***/
/*****************************************************************************************************/
HRESULT faceTrackerImp::play(HANDLE handleCamera)
{
    HRESULT hResult = S_OK ;

    CameraFT *pCameraFT = (CameraFT *) handleCamera ;

    hResult = pCameraFT->pMediaControl->Run() ;

    return hResult ;
}
```

Une fonction stopAndQuit() permet de stopper le rendu de flux vidéo et quitter « proprement » tout les appels à ce bloc DirectShow, en libérant notamment la mémoire allouée au processus et en supprimant les filtres et interfaces créées les uns après les autres :

 : « STOP » pour arrêter les actions sur le flux de donner et le stopper

Une fonction `ForegroundPreview`() permet de forcer la Video Window à s'afficher sur un « top layer ». :
(Cette fonctionnalité a été implémentée mais n'a pas été utilisée dans le projet, car elle « abuse » le système en forçant la fenêtre à s'afficher)

Une fonction `takePhoto()` permet de « voler » une image destiné au « streaming vidéo » à l'aide d'un trigger vidéo implémenté sur le filtre de CaptureVideo. Ce trigger est une interface de type `IAMVideoControl` `pVideoControl`.
Remarque :
Quelques explications afin de mieux comprendre le schéma qui suit
Le nouvel élément suivant représente :

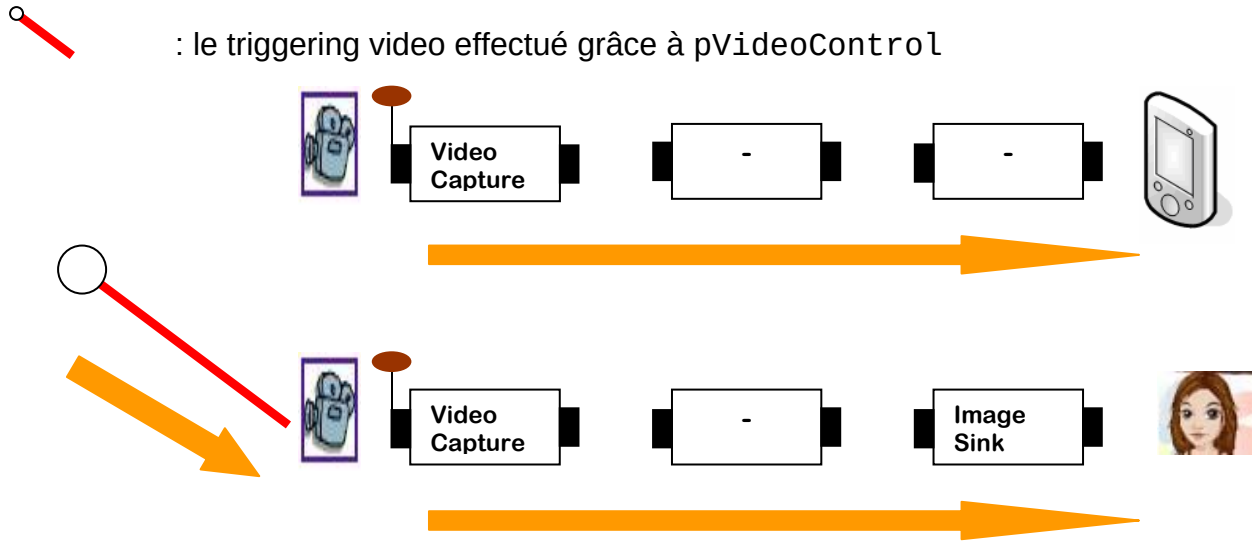
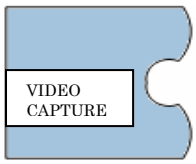


Figure 53 : Fonctionnement du triggering pour prendre une image du flux

Le triggering qui permet de changer le chemin des données vers la création d'un fichier ne se fait que dans la fonction `takePhoto()`, sinon dans le cas normal le chemin des données est dirigé vers le « window preview ».

Création de la DLL avec les fonctions à exporter

Notre bloc regroupant tout les éléments nécessaires à la capture vidéo est maintenant complet.



Les fonctions de la « pièce de puzzle » ci-dessus sont prêtes à être exportée.

Nous allons pour cela implémenter deux nouveaux fichiers permettant l'exportation claire et concise des fonctions désirées.

Les fichiers sont les suivants :

- ExportedFunctionsFromDLL.h
- ExportedFunctionsFromDLL.cpp

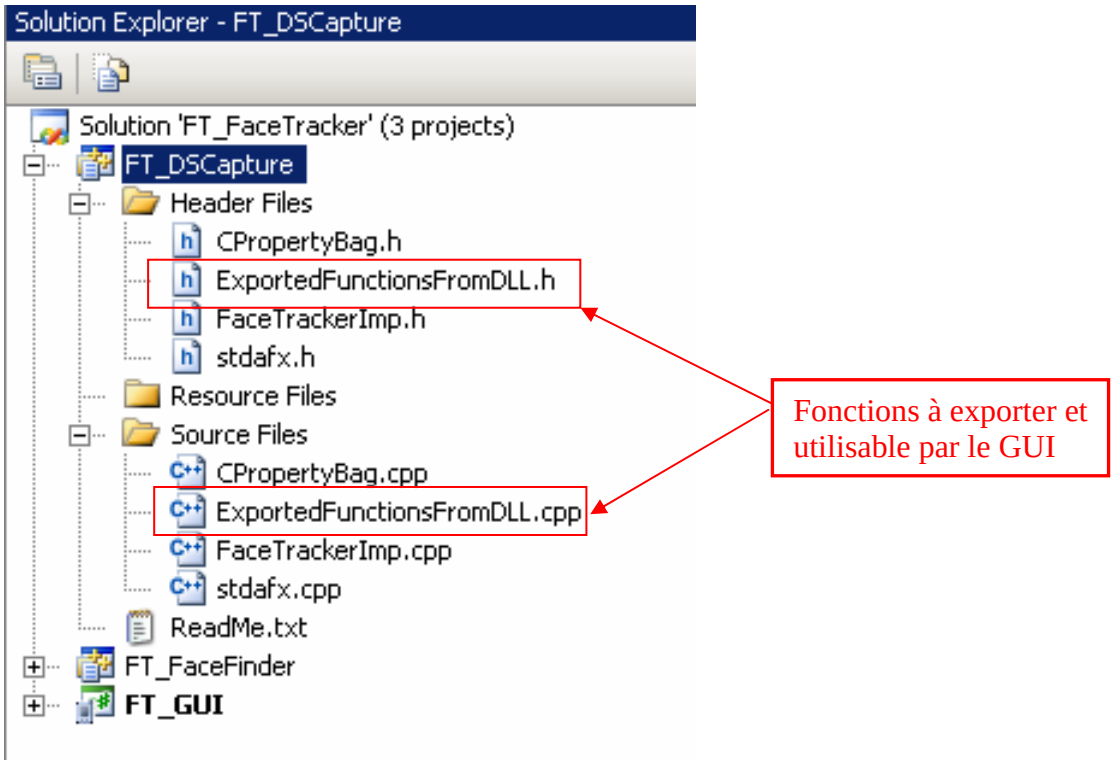


Figure 54 : Project Solution

La pièce maîtresse du puzzle est le GUI. Elle représente donc un bloc qui aimerait simplement faire appel aux fonctionnalités de DirectShow sans pour autant aller

chercher les spécificités du code de DirectShow tels que la détection du driver de la caméra du PDA. En effet, toutes ces fonctionnalités internes doivent être caché.

Dans ce cas de figure, nous allons exposer dans le fichier ExportedFunctionsFromDLL.cpp 4 fonctions très simples pour le GUI, à savoir :

- previewDisplay(HWND pbhandle)
- updatePreviewNow()
- stop()
- takePicture()

la fonction previewDisplay(HWND pbhandle) prend en paramètre un « handle » sur un pictureBox(fenêtre de dessin) passé par le GUI.
Il s'agit là de la fonction la plus importante, car elle se charge de faire les connexions nécessaires et de démarrer la capture vidéo.

Problème :

Cependant pour démarrer une capture vidéo, nous sommes obligés de lancer la capture et de créer une boucle infinie immédiatement après permettant ainsi de continuer le « preview ».
En effet, une « fenêtre de preview » à l'écran signifie bien que l'application continue de tourner.

Alors comment faire appel aux autres fonctions sans arrêter le « preview du flux vidéo »?

De plus l'appel aux autres fonctions va poser problème si nous passons le même « handle » du pictureBox. Cela va incontestablement amener l'application à planter.

Solution :

Un petit gestionnaire interne d'évènements a été implémenté au cœur de ce fichier. Ainsi, le GUI va continuer à appeler normalement les fonctions exposées par cette DLL. De plus la première fonction appelée par le GUI est previewDisplay(HWND pbhandle)

L'astuce se trouve à deux niveaux :

- au niveau des paramètres des 4 fonctions exposées ci-dessus. En effet, nous remarquons que mis à part la fonction previewDisplay(HWND pbhandle), les autres fonctions ne possèdent pas de « handle » sur le pictureBox passé par le GUI (donc l'application ne peut pas planter)
- au niveau d'un gestionnaire d'évènements permettant de faire appel aux autres fonctions depuis la boucle infinie de la fonction previewDisplay(HWND pbhandle)

Pour plus d''informations sur les fonctions et fichiers développés, prière de voir les annexes et le code source.

Détection de visage

FACE TRACKING

Nous allons à présent voir l'implémentation de l'autre DLL qui englobe la libraire de l'IDIAP qui permet de faire de la détection de visage. Cette DLL va ensuite comme tout à l'heure exposer des fonctions qui peuvent être accédée par la « pièce maîtresse GUI ».

Il existe 2 versions de cette DLL. :

- possédant au préalable les anciens algorithmes de l'IDIAP (Torch3Vision basée sur la lilbrairie Torch) et qui a été utilisé pour le projet Sabbuca. Cette librairie a été portée sur Windows Mobile 5.0 -6.0
- la nouvelle librairie « FaceFindingAPI_0.2.2 » (également basée sur la librairie Torch) a été portée sur Windows Mobile 5.0 - 6.0

Dans l'application Face Tracker, seul la deuxième version avec les nouveaux algorithmes de l'IDIAP été utilisée.

Toutefois les deux versions ont été portées dans une DLL avec un GUI approprié dans le cadre d'un démonstrateur réalisé pour un autre projet : « MemoriaMea » (plus particulièrement son sous-projet « Facia Mea »)

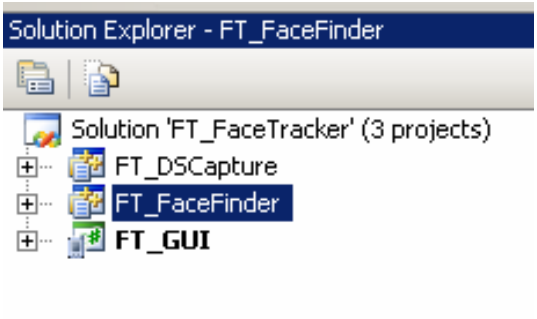


Figure 55 : Solution Explorer

Ci-dessus, nous pouvons voir dans la *solution* 'FT_FaceTracker' avec un projet « FT_FaceFinder ». C'est ce projet qui englobe tout les fichiers nous permettant de faire de la détection de visage. L'output de ce projet est comme expliqué plus haut, une DLL nous permettant de faire de la détection de visage sur une image.

Nous remarquons dans les images ci-dessous que le projet contient un certain nombre de fichiers, qui sont les fichiers développés par l'IDIAP et portés pour le système Windows Mobile 5.0 et 6.0 à l'exception des fichiers

`ExportedFunctionsFromDLL.h` et `ExportedFunctionsFromDLL.cpp` qui permettent d'exporter des fonctions facilement utilisable par le GUI

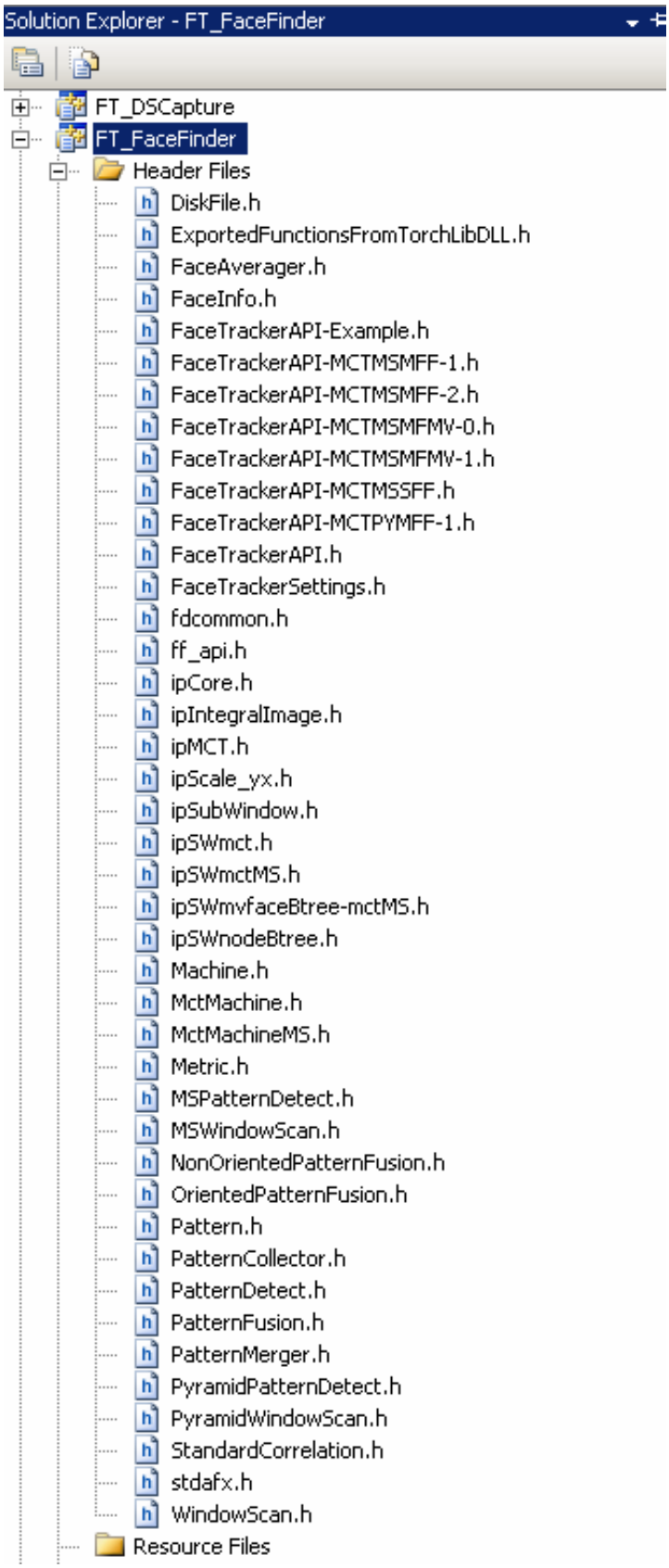


Figure 56 : Headers Files dans le projet

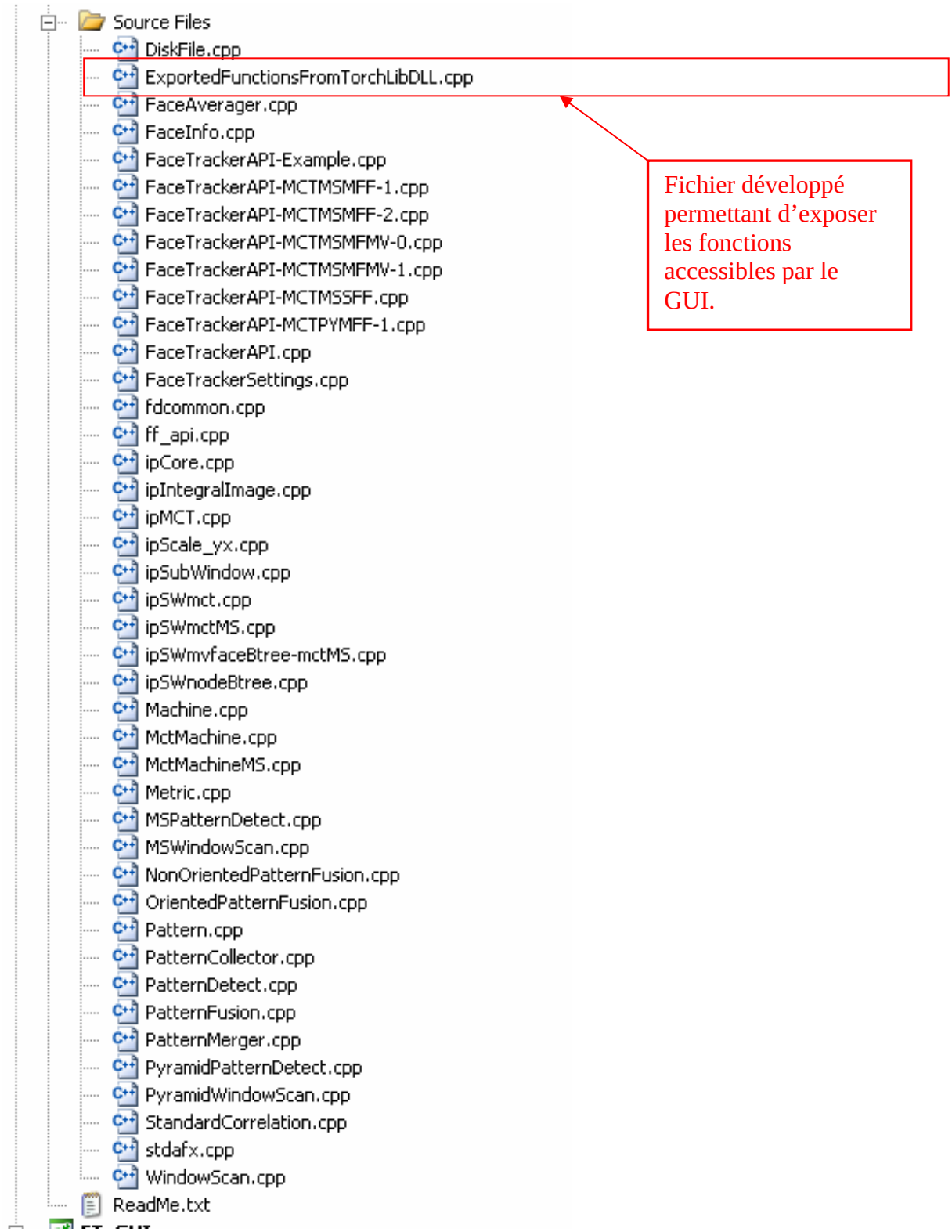


Figure 57 : Source Files dans le projet

Librairie FaceFindingAPI_0.2.2

Introduction

Il s'agit de la nouvelle libraire (en date du 21 novembre 2007) fournit par l'IDIAP (institut de recherche spécialisé dans les interfaces homme machine multimodales).
Il s'agit d'une librairie contenant les algorithmes nécessaires à la détection de visage dans une image.

Fonctionnement de la librairie

La libraire demande en entrée :

- bien naturellement l'image sur laquelle elle doit trouver s'il y a un visage ou non. Sur ce fichier image, la librairie va effectuer un dégradé de gris des pixels (grayscalling) qui vont ensuite être passés à l'algorithme pour la détection de visage
- un fichier « .model » contenant le model de détection de visage. Ce fichier spécifie les informations relatives à la détection d'un visage « universel »(pour ne pas dire quelconque). Le fichier utilisé se nomme « `frontal.model` »

Cas d'utilisations :

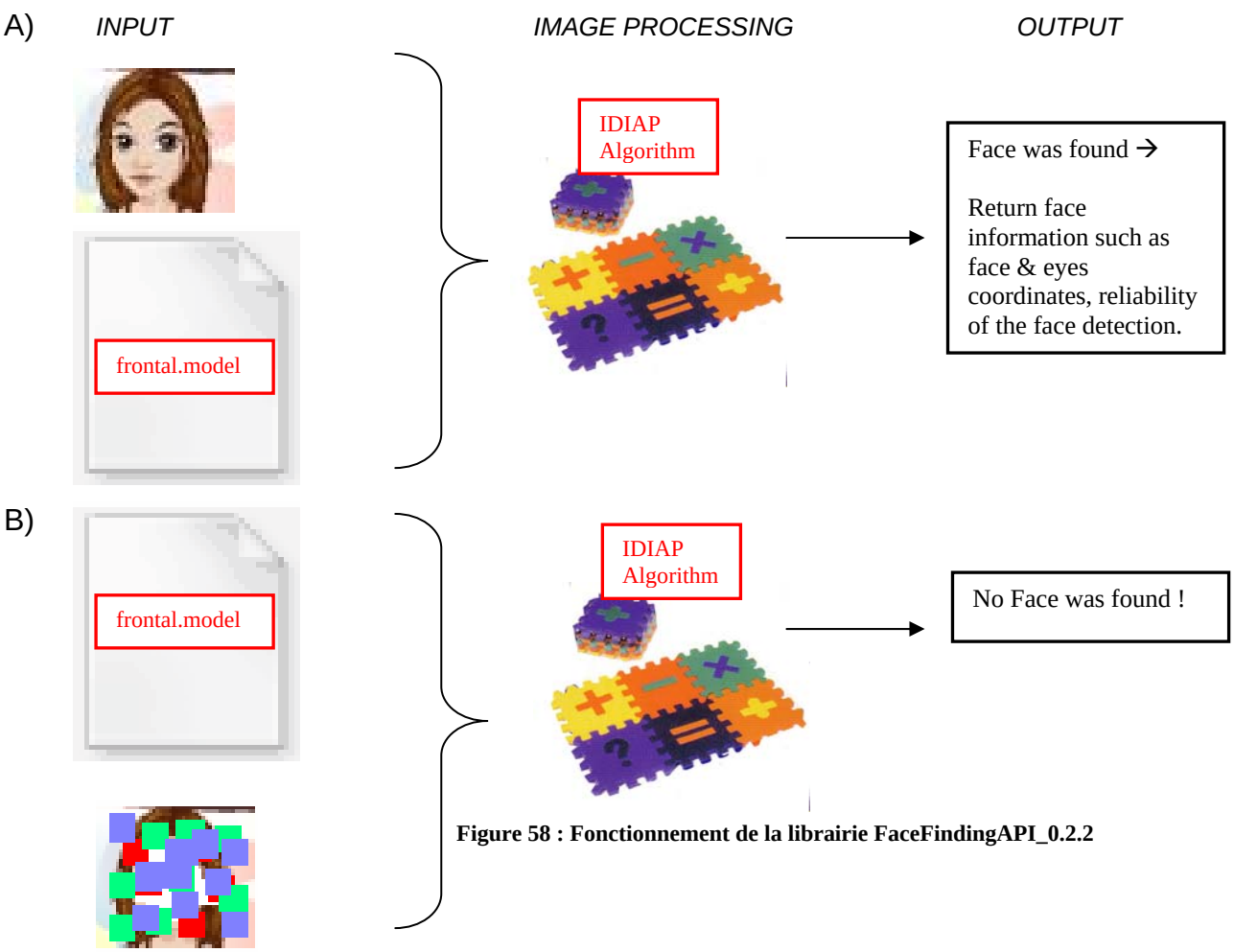


Figure 58 : Fonctionnement de la librairie FaceFindingAPI_0.2.2

Les fichiers image que nous passons à notre algorithme sont toujours des fichiers .bmp RGB 24 bits.

Afin de pouvoir comprendre comment traiter un tel fichier, nous allons voir comment s'organise ce dernier.

Structure d'un fichier BITMAP

Le format Bitmap (BMP), est un format graphique qui a été étudié dans le but de ne dépendre d'aucun type de périphérique particulier, d'où la dénomination de Device-Independent Bitmap (DIB) pour ce format. Les données de l'image que sont les pixels sont stockées sous forme de tableau.

Un fichier bitmap s'organise de la façon suivante :

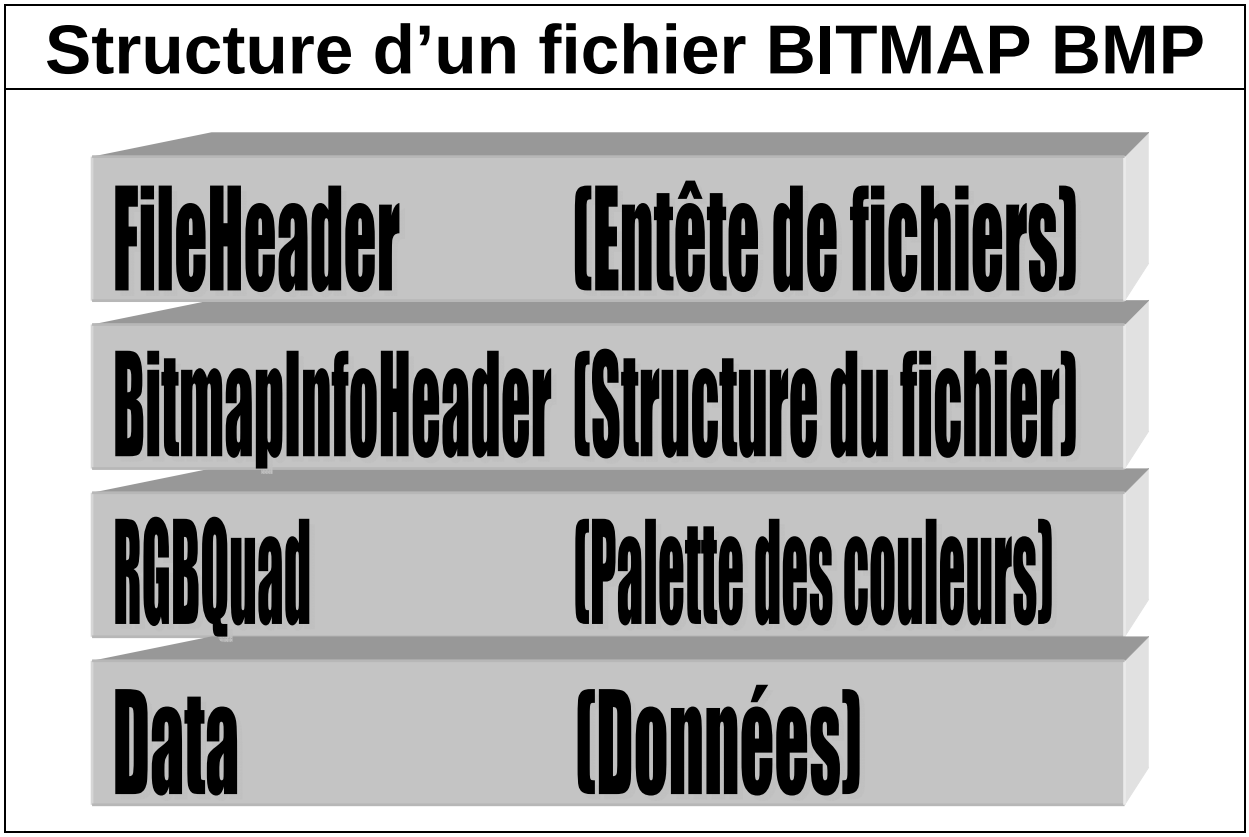


Figure 59 : Structure d'un fichier BMP

Ces derniers sont atteignables en C++ de la manière suivante :

```
BITMAPFILEHEADER    bmfh;
BITMAPINFOHEADER    bmih;
RGBQUAD             aColors[];
BYTE                aBitmapBits[];
```

Exemple tiré du site http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html en date du 22 novembre 2007

Les structures complètes des deux premiers blocs sont les suivantes :
Les 2 tableaux qui suivents sont tiré du site
http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html en date du 22 novembre 2007.

The BITMAPFILEHEADER:

start	size	name	stdvalue	purpose
1	2	bfType	19778	must always be set to 'BM' to declare that this is a .bmp-file.
3	4	bfSize	??	specifies the size of the file in bytes.
7	2	bfReserved1	0	must always be set to zero.
9	2	bfReserved2	0	must always be set to zero.
11	4	bfOffBits	1078	specifies the offset from the beginning of the file to the bitmap data.

The BITMAPINFOHEADER:

start	size	name	stdvalue	purpose
15	4	biSize	40	specifies the size of the BITMAPINFOHEADER structure, in bytes.
19	4	biWidth	100	specifies the width of the image, in pixels.
23	4	biHeight	100	specifies the height of the image, in pixels.
27	2	biPlanes	1	specifies the number of planes of the target device, must be set to zero.
29	2	biBitCount	8	specifies the number of bits per pixel.
31	4	biCompression	0	Specifies the type of compression, usually set to zero (no compression).
35	4	biSizeImage	0	specifies the size of the image data, in bytes. If there is no compression, it is valid to set this member to zero.
39	4	biXPelsPerMeter	0	specifies the the horizontal pixels per meter on the designated targer device, usually set to zero.
43	4	biYPelsPerMeter	0	specifies the the vertical pixels per meter on the designated targer device, usually set to zero.
47	4	biClrUsed	0	specifies the number of colors used in the bitmap, if set to zero the number of colors is calculated using the biBitCount member.
51	4	biClrImportant	0	specifies the number of color that are 'important' for the bitmap, if set to zero, all colors are important.

Il y a trois données importantes à retenir concernant le BITMAPINFOHEADER :

start	size	name	stdvalue	purpose
19	4	biWidth	100	specifies the width of the image, in pixels.
23	4	biHeight	100	specifies the height of the image, in pixels.
29	2	biBitCount	8	specifies the number of bits per pixel.

La taille de l'image en pixels est spécifiée grâce aux paramètres :

- `biWidth`
- `biHeight`

Etant donné que nous ne traitons que les fichiers BMP 24 bits RGB, nous pouvons vérifier le nombre de bits par pixels utilisé à l'aide du paramètre

- `biBitCount`

L'appel à ces paramètres se fera très simplement en C++ de la manière suivante :

```
bmih.biBitCount
bmih.biWidth
bmih.biHeight
```

Il faudra donc d'une manière ou d'une autre, toujours s'assurer de transmettre une image BMP RGB 24 bits étant donné que nous ne traitons que ce cas là!

Le bloc RGBQUAD représente la tables des couleurs : en mode 1 bit, par exemple, elle ne contiendra que deux entrées (noir/blanc)

Le tableau qui suit est tiré du site http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html en date du 22 novembre 2007.

The RGBQUAD array:

The following table shows a single RGBQUAD structure:

start	size	name	stdvalue	Purpose
1	1	rgbBlue	-	specifies the blue part of the color.
2	1	rgbGreen	-	specifies the green part of the color.
3	1	rgbRed	-	specifies the red part of the color.
4	1	rgbReserved	-	must always be set to zero.

Enfin il est important de préciser qu'il y a une différence entre l'image .BMP telle que nous la voyons s'afficher et la manière dont les données sont stockés dans le fichier.

En effet l'image vue à l'écran est stockée de façon inversée dans le fichier. Il y a une symétrie d'axe horizontale (flip vertical).

Etant donné que nous sommes sur un codage RGB 24 bits, alors cela signifie que chaque pixel est codé sur 3 bytes (3 bytes * 8 = 24 bits). Etant donné que chaque pixel est composé des couleurs R (Red – rouge), G (Green – vert), B (Blue – bleu), cela signifie que chacune des couleurs est codée sur un byte.
Nous devons donc lire le fichier dans le sens opposé si nous voulons récupérer des données, il faut donc faire attention au fait que l'ordre dans lequel nous lisons n'est plus RGB mais BGR.

Image vu à l'écran *Image stockée sur le fichier*

Figure 60 : Images telles que vues à l'écran et stockés dans un fichier BMP

Portage de la librairie sur Windows Mobile

Le grand défit a été de porter cette librairie sur un système *MS Windows Mobile*.

Il faut savoir que tous les fichiers de cette librairie ont été intégrés dans le portage du Face Tracker sur le système Windows Mobile.

Cependant, un certain nombre de changements ont dû être effectué dans le code de la librairie « FaceFindingAPI_0.2.2 ».

Ce chapitre regroupe toutes les modifications apportées au code fournis par l'IDIAP en montrant à chaque fois le code original et les modifications apportées.

Modification des fichiers d'en-tête (Header Files)

Fichier `ff_api.h` :

Contrairement à l'ancienne librairie, cette nouvelle librairie n'exposait pas les fonctions spécifiant les coordonnées des yeux, il a donc fallut les exposer :

La position des yeux sur une image est spécifiée suivant les coordonnées (x,y) en pixels.

Le code suivant a été rajouté :

Modifications apportées :
Lignes 118 à 123 :

```
//eye information
int xeyel;
int yeyel;
int xeyer;
int yeyer;
```

Modification des fichiers sources (Source Files) :

Fichier `ff_api.cpp` :

Contrairement à l'ancienne librairie, cette nouvelle librairie n'exposait pas les fonctions spécifiant les coordonnées des yeux, il a donc fallut les exposer :

La position des yeux sur une image est spécifiée suivant les coordonnées (x,y) en pixels.

Le code suivant a été rajouté :

Modifications apportées :
Ligne 570 à 575 :

```
faceinfo->xeyel= finder->tracker->faces[number]->eyeL_x ;
faceinfo->yeyel= finder->tracker->faces[number]->eyeL_y  ;
faceinfo->xeyer = finder->tracker->faces[number]->eyeR_x ;
faceinfo->yeyer= finder->tracker->faces[number]->eyeR_y ;
```

Pour plus d''informations sur les changements amenés à « FaceFindingAPI_0.2.2 » voir les annexes et le code source.

Intégration dans le projet

Comme pour le bloc de capture vidéo, ce bloc sera intégré dans une DLL. Cette dernière va exporter des fonctions qui pourront être appelé depuis le GUI.

Ici, une seule fonction est exportée, il s'agit de la fonction suivante :

```
track(  char * worldModelPath,
        char * fileNameToTrack,
        char grayMode,
        _ff_faceinfo_t** faceinfo,
        float score )
```

Cette fonction prend en paramètre :

- le chemin vers le fichier model `frontal.model` (ou plutôt le lien vers dossier dans lequel se trouve le fichier)
- le chemin vers le fichier image .bmp à tester pour la détection de visage
- le type de grayscalling (dégradé de gris) à effectué sur l'image dans le cas où il y a plusieurs types de grayscale
- un pointeur vers une structure représentant le visage (en effet la fonction va permettre de mettre a jour les coordonnées de cette structure qui comprends les coordonnées (x,y) des yeux et du visage de la personne)
- un « score » spécifiant fiabilité du résultat de la détection de visage (dans le cas de l'implémentation d'un futur système d'authentification, ce « score » va jouer un rôle plus important)

Cependant cette fonction track(…), exportée pour le GUI, va faire appel à une autre fonction interne qui va permettre de charger l'image en mémoire:

```
-loadbmp( char *filename,
          int *image_width,
          int *image_height,
          unsigned char **image)
```

Cette fonction fait un passage de paramètre par référence avec :

- le chemin vers le fichier image .bmp à tester pour la détection de visage
- la largeur de l'image
- la hauteur de l'image
- un pointeur sur l'image à traiter afin d'allouer l'espace mémoire nécessaire à l'image

Explications sur la fonction loadbmp (…) :

Les commentaires sont normalement assez explicites mais pour aller voir les parties importantes de ce code, prière de voir les Annexes.

Explications sur la fonction track (…) :

Les commentaires sont normalement assez explicites mais pour aller voir les parties importantes de ce code, prière de voir les Annexes.

Compilation et déploiement de l'algorithme

La compilation et le déploiement de l'algorithme dans le dossier n'ont pas posé de problème.

Résultat *:*

L'algorithme prend en moyenne 1 à 2 secondes avec les optimisations du détecteur pour traiter une image .bmp sur Windows Mobile 5.0 et 6.0.

Il faudrait utiliser un outil comme « ThreadX » (sous Windows XP) permettant de regarder plus en détail les endroits où nous perdons du temps. De plus un autre changement des paramètres du détecteur pourrait peut-être amené une plus grande rapidité

Néanmoins il faut admettre que ce temps est acceptable dans la mesure où actuellement (21 novembre 2007), les PDA ne sont pas équipés de FPU (Floating Point Unit) qui est l'unité spécialement dédiée aux calculs sur des nombres à virgule flottante. Il y a donc une émulation du FPU qui est faite par le processeur.

Pour rappel, les PDA sont en général tous basé sur un processeur ARM. Les concepteurs de processeurs ARM ont décidé d'intégrer une FPU directement sur le cœur pour les AMR10E et plus. Ils ont en effet remarqué que les floating points sont de plus en plus utilisés, alors au lieu de considérer un coprocesseur séparé, ils l'intègrent déjà dans les nouveaux cœurs. Le problème est que les premiers PDA disposant de ce processeur ne sont pas disponibles actuellement encore (21 novembre 2007).

Nous allons maintenant voir le portage de l'ancienne librairie qui a posé un peu plus de problème que la nouvelle.

Portage de la librairie Torch3Vision de l'IDIAP (ancienne librairie –version telle qu'utilisée dans le projet SABBUCA (version février 2007 de Michael Clausen))

Librairie Torch3 Introduction

La librairie « Torch » de l'IDIAP, écrite dans le langage C++, comprend un certain nombre de classe dont celles nécessaires pour le traitement de la voix et de l'image.
La version de Torch utilisée pour le projet « Face Tracker on Windows Mobile » est « Torch3vision ». Une description détaillée est donnée sur le lien suivant : http://torch3vision.idiap.ch .
« Torch3vision » est une librairie permettant la « vision machine », à savoir, interpréter et détecter des visages sur une photo par exemple. « Torch3vision » est basé sur la librairie « Torch » qui permet l'apprentissage machine (machine-learning) dans son sens le plus large, à savoir, elle englobe tout ou une partie des algorithmes tels Bagging, AdaBoost, modèles de Markov, réseaux neuronaux,…

Fonctionnement de la librairie

La libraire demande en entrée :

- bien naturellement l'image sur laquelle elle doit trouver s'il y a un visage ou non. Cependant l'image doit être passé pixels par pixels et en dégradé de gris contrairement à la nouvelle version de l'algorithme ou ce « grayscalling » est

interne à l'algorithme. Sur ce « grayscale » de l'image, la librairie va pouvoir lancer détection de visage
- un fichier « .vwm » contenant le model de détection de visage. Ce fichier spécifie les informations relatives à la détection d'un visage « universel »(pour ne pas dire quelconque). Le fichier utilisé se nomme « faceDetection.vwm »

Cas d'utilisations :

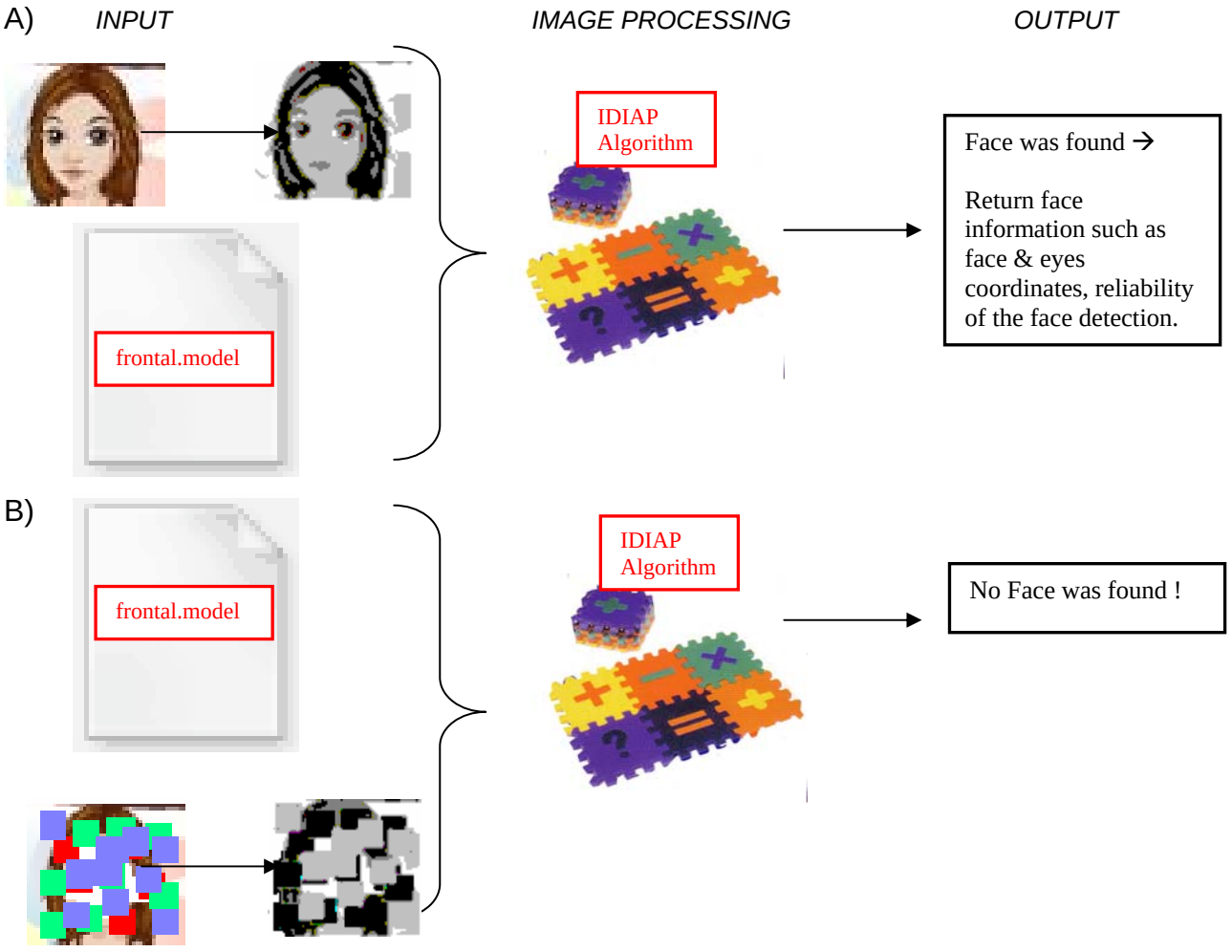


Figure 61 : Fonctionnement de l'ancienne librairie Torch3Vision (en date de janvier 2007)

Portage de la librairie sur Windows Mobile

La librairie Torch3vision ayant pu être utilisée et testée de façon correcte sur les systèmes suivants *Linux, SunOS, FreeBSD, OSF1, Mac OS X and even MS Windows*. Le grand défit a été de le porter sur un système *MS Windows Mobile*.

A cet effet, un certain nombre de changements a dû être effectué dans le code de la librairie « Torch3vision ».

Fichiers de la librairie Torch à intégrer au projet Face Tracker sur Windows Mobile

Tous les fichiers de la librairie de l'IDIAP ne sont pas utiles dans l'intégration du Face Tracker sur le système Windows Mobile. Seuls ceux permettant d'initialiser et de lancer le détecteur et de scanner une image à la recherche d'un visage sont utiles.

Torch3Vision est organisé en plusieurs répertoires :

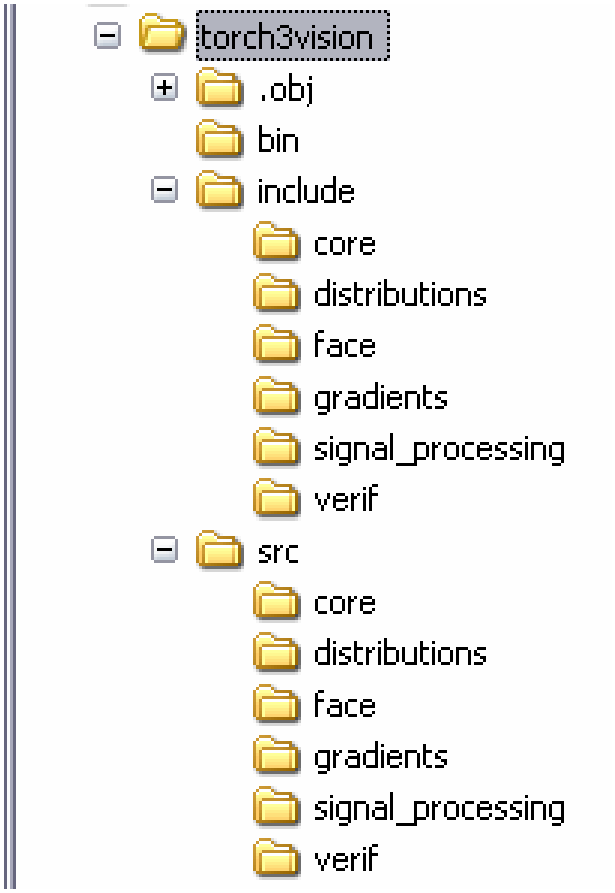


Figure 62 : Organisation de Torch3Vision

Fichiers d'en-tête (Include) de la librairie « Torch » à intégrer

core

Allocator.h
Dataset.h
DiskXFile.h
general.h
List.h
Machine.h
Xfile.h
Sequence.h
Stack.h
string_utils.h
Object.h
PreProcessing.h

Figure 63 : Fichiers .h à intégrer du dossier core

Figure 64 : pas de fichier à intégrer du dossier distributions

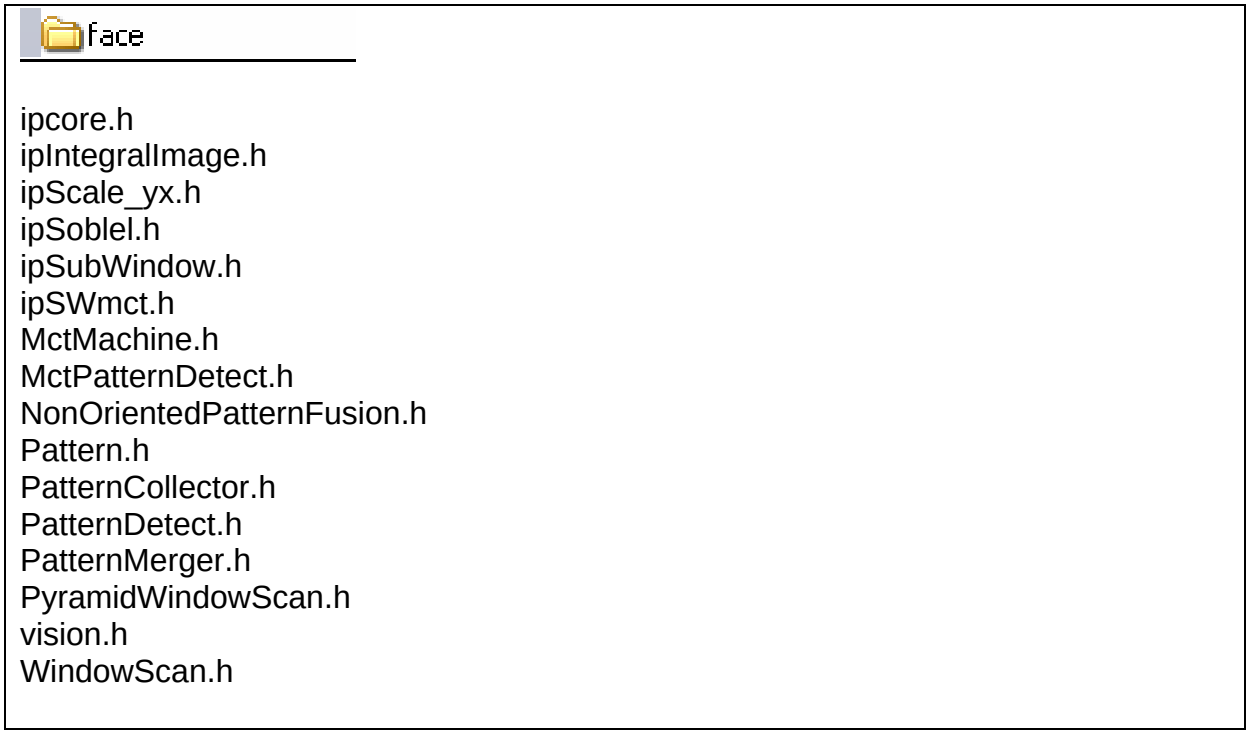


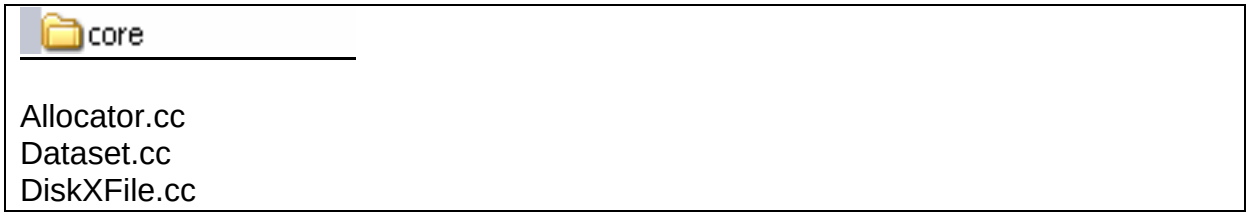
Figure 65 : fichiers .h à intégrer du dossier face

Figure 66 : pas de fichier à intégrer du dossier gradients

Figure 67 : pas de fichiers à intégrer du dossier signal_processing

Figure 68 : Fichier .h à intéger du dossier verif

Fichiers sources de la librairie « Torch » à intégrer



general.cc
Machine.cc
Xfile.cc
Sequence.cc
Stack.cc
string_utils.cc
Object.cc
PreProcessing.cc

Figure 69 : Fichiers .cc à intégrer du dossier core

face

ipcore.cc
ipIntegralImage.cc
ipScale_yx.cc
ipSoblel.cc
ipSubWindow.cc
ipSWmct.cc
MctMachine.cc
MctPatternDetect.cc
NonOrientedPatternFusion.cc
Pattern.cc
PatternCollector.cc
PatternDetect.cc
PatternMerger.cc
PyramidWindowScan.cc
vision.cc
WindowScan.cc

Figure 70 : Fichiers .cc à intégrer du dossier face

gradients

-

Figure 71 : pas de fichiers à intégrer du dossier gradients

signal_processing

-

Figure 72 : pas de fichiers à intégrer du dossier signal_processing

verif

FaceDetector.cc

Figure 73 : Fichier .cc à intégrer du dossier verif

Modifications apportés dans les fichiers cités plus haut :

Le grand défit a été de porter cette librairie sur un système *MS Windows Mobile*.

Il faut savoir que tous les fichiers n'ont pas été intégrés dans le portage du Face Tracker sur le système Windows Mobile.

Cependant, un certain nombre de changements ont dû être effectué dans le code de la librairie « Torch3Vision ».

Remarque : cette librairie contrairement à la nouvelle exposait déjà les coordonnées des yeux

Pour plus d''informations sur les fonctions et fichiers développés, prière de voir les annexes et le code source.

Intégration dans le projet

Comme pour le bloc de capture vidéo, ce bloc sera intégré dans une DLL. Cette dernière va exporter des fonctions qui pourront être appelé depuis le GUI.

Il a fallut créer un wrapper autour de l'ancienne librairie. Prière de voir les commentaires du code pour plus de précision.

Explications détaillées du code

Il a néanmois été jugé intéressant de mettre le code du wrapper dans ce rapport avec explications.

(Remarque : Si vous ne souhaitez pas lire le code, vous pouvez passer directement à la page 106-108 pour la suite)

Fichier WrappedFaceDetector.h

```
/*****************************************************************************************************/
/*!
 *  \file
 *      WrappedFaceDetector.h in TorchPortageDLL -
 *  FACETRACKER ON WINDOWS MOBILE BASED PHONE / This header file includes IDIAP's FaceDetector
 *                                             It creates a wrapper around the Torch Library
 *
 *  \date       October 2007
 *  \author     Flavio Tarsetti (flavio dot tarsetti at mycable dot ch)
 *                              (tarsflav at students dot hevs dot ch )
 *                              flavio.tarsetti@mycable.ch  or tarsflav@students.hevs.ch
 */
/*****************************************************************************************************/

/*** Including header files to be used  ***************************************************************/
#include "FaceDetector.h"
#include "stdafx.h"

#include <stdlib.h>


/*** Torch includes & prototypes **********************************************************************/
namespace Torch
{
    class FaceDetector;
}
/*****************************************************************************************************/


class WrappedFaceDetector
{
    private :

        bool _newFaceAvailable ;
        uint _movementSensitivity ;
        int _width ;
        int _height ;
        BOOL success ;


    public  :
        Torch::FaceDetector *_pDetector ;
        //Constructors
        WrappedFaceDetector();
        WrappedFaceDetector(char* worldModelPath, char *fileName, char grayMode) ;

        //Destructor
        virtual ~WrappedFaceDetector() ;

        //Functions
        const int getMovementSensitivity() const ;

        void setMovementSensitivity(const uint sensitivity) ;

        Sequence* loadBitmapFromFile(char* fileName , Sequence *buffer);

        //mutators
        void setWidth(int width) ;
        void setHeight(int height) ;

        //accessors
        int getWidth() ;
        int getHeight() ;
        BOOL getSuccess() ;

} ;
```

Constructeurs :

Le constructeur utilisé pour englober les fonctions de « Torch » est le deuxième sur l'image ci-dessous (le premier étant simplement le constructeur par défaut).

Il prend 3 paramètres, à savoir :

- le chemin vers le fichier représentant le worldmodel (modèle de visage). Le worldModel utilisé est celui fourni par l'IDIAP et se nomme `FaceDetection.vwm`.
- le chemin vers le fichier .bmp où se trouve l'image avec un visage à détecter (où pas…)
- le type de dégradé de gris (grayscale) à utiliser

```
//Constructors
WrappedFaceDetector();
WrappedFaceDetector(char* worldModelPath, char *fileName, char grayMode) ;
```

Destructeur :

Ce dernier se charge de détruire le détecteur qui a été créé

```
//Destructor
virtual ~WrappedFaceDetector() ;
```

Membres publics et privés :

Membres privés

- Nous remarquons la variable privée booléenne `success` qui représente le résultat de la détection de visage. Si `success` vaut « 1 » alors cela signifie qu'un visage a été détecté sur l'image, sinon elle vaudra « 0 » et cela signifie qu'aucun visage n'a pu être détecté
- Les variables privées _width et _height sont initialisées lors de la lecture de l'image à traiter et contiennent les informations sur la taille de l'image. Elles sont utiles car elles permettent également d'initialiser le détecteur sur l'image à analyser.

```
private :

    bool _newFaceAvailable ;
    uint _movementSensitivity ;
    int _width ;
    int _height ;
    BOOL success ;
```

Résultat de la détection de visage

Membres publics :

- `_pDetector` est un pointeur sur un objet de type FaceDetector. Il représente le détecteur de visage.
- La fonction `getSuccess()` retourne un booléen qui est la variable privée `success` représentant la détection d'un visage ou pas.

- setWidth() et setHeight permettent d'initialiser les variables privées _width et respectivement _height.
- getWidth() et getHeight() permettent de récupérer les variables privées _width et respectivement _height()
- La fonction LoadBitmapFromFile prend en paramètre le nom du fichier image à analyser et un pointeur vers un objet de type Sequence (définit dans la librairie Torch. Cette fonction va se charger de lire l'image byte par byte et d'en faire un dégradé de gris (grayscale) sur les pixels de l'image. Les pixels en niveau de gris sont ensuite enregistrés vers le tableau buffer de type Sequence. Ce tableau de pixels est ensuite retourné par la fonction. C'est ce tableau de type Sequence qui sera passé au détecteur de visage.

```
public   :
    Torch::FaceDetector *_pDetector ;
    //Constructors
    WrappedFaceDetector();
    WrappedFaceDetector(char* worldModelPath, char *fileName, char grayMode) ;

    //Destructor
    virtual ~WrappedFaceDetector() ;

    //Functions
    const int getMovementSensitivity() const ;

    void setMovementSensitivity(const uint sensitivity) ;

    Sequence* loadBitmapFromFile(char* fileName , Sequence *buffer);

    //mutators
    void setWidth(int width) ;
    void setHeight(int height) ;

    //accessors
    int getWidth() ;
    int getHeight() ;
    BOOL getSuccess() ;
```

L'image à analyser est passée pixel par pixel à un tableau de type Sequence

Fichier WrappedFaceDetector.cpp :

```
/*****************************************************************************************/
/*!
 *  \file
 *      WrappedFaceDetector.cpp in TorchPortageDLL -
 *  FACETRACKER ON WINDOWS MOBILE BASED PHONE / This file includes IDIAP's FaceDetector
 *                                              It implements a wrapper around the Torch Library
 *
 *  \date       October 2007
 *  \author     Flavio Tarsetti (flavio dot tarsetti at mycable dot ch)
 *                              (tarsflav at students dot hevs dot ch )
 *                              flavio.tarsetti@mycable.ch  or tarsflav@students.hevs.ch
 */
/*****************************************************************************************/

/*** Including header files to be used  ***************************************************/
#include "FaceDetector.h"
#include "WrappedFaceDetector.h"

/*** global variable Declaration **********************************************************/
const uint DefaultMovementSensitivity = 10 ;


/*** Constructors *************************************************************************/
WrappedFaceDetector::WrappedFaceDetector()
{
    //default constructor
}

WrappedFaceDetector::WrappedFaceDetector(char *worldModelPath,char *fileName,char grayMode)
: _pDetector(NULL), _newFaceAvailable(false) , _movementSensitivity(DefaultMovementSensitivity)
{

    //  Check if the WorldModelFile exists
/*  if(fopen(worldModelPath,"r")==0)
    {
        MessageBox (NULL, _T("ERROR:WorldModelFile doesn't exist!%c",worldModelPath),
                    _T("ERROR"), MB_OK);

    }
    else
    {*/
        //buffer is a Torch Sequence object that stores the grayscale pixel from the image
        //for the face tracking's algorithm
        Sequence *buffer;

        //file is loaded, grayscaled and stored in buffer object
        buffer = loadBitmapFromFile(fileName, buffer) ;


        //initialize detector
        _pDetector = new Torch::FaceDetector((char *)worldModelPath , getWidth() , getHeight()) ;

        //scan buffer & hope we find a face... ;)
        success = (_pDetector->scan(buffer)>-FLT_MAX);


        //_pDetector->~FaceDetector();


        _pDetector->allocator->freeAll();


    /*}*/


}
/*** Destructor ***************************************************************************/
WrappedFaceDetector::~WrappedFaceDetector()
{
    //Free memory
    if(_pDetector)
    {

        delete _pDetector;


    }

}

/*****************************************************************************************/
/*** Function loadBitmapFromFile(char *fileName, Sequence *buffer)*************************/
/*****************************************************************************************/
/*** This function is used to grayscale an bmp image in order to prepare Torch's Sequence      ***/
/*** object for face tracking                                                                   ***/
/***                                                                                           ***/
/*** 2 parameters : 1) the image to track the face                                              ***/
/***                2) the buffer to store the graycaled pixel                                 ***/
/*** return value : pointer to a Sequence object (Torch Library object                         ***/
/*****************************************************************************************/
Sequence* WrappedFaceDetector::loadBitmapFromFile(char *fileName, Sequence *buffer)
{
    //Local Declarations
    char *pStr1 = fileName;

    FILE *myFile ;
    //Bitmap sections
    BITMAPFILEHEADER bmfh ;
    BITMAPINFOHEADER bmih ;

    //Opening file
    if((myFile = fopen(pStr1, "rb"))==NULL){
            MessageBox (NULL, _T("file not opened"),
                        _T("ERROR"), MB_OK);
        printf( "The file with the face to be tracked was not opened\n" ) ;
    }else{

            printf ("The file with the face to be tracked is opened\n") ;}
    //check if file is opened...
    if(myFile==NULL)
    {
        MessageBox (NULL, _T("ERROR:File not opened!"),
                    _T("ERROR"), MB_OK);
    }

    //Reading information header & file header
    fread(&bmfh , sizeof(BITMAPFILEHEADER) , 1 , myFile) ;
    fread(&bmih , sizeof(BITMAPINFOHEADER) , 1 , myFile) ;

    printf("Bits per pixel = %d\n", bmih.biBitCount) ;
    //Is the BMPFile 24 bits ? (To test with device in release mode)
    /*if(bmih.biBitCount == 24)
    {
        MessageBox (NULL, _T("BMPFile : 24 Bits"),
                    _T("INFORMATION"), MB_OK);
    }
    else
    {
        MessageBox (NULL, _T("BMPFile : not a 24 Bits"),
                    _T("INFORMATION"), MB_OK);
    }
    */
```

```
    //set width& height
    setHeight(bmih.biHeight) ;
    setWidth(bmih.biWidth)   ;

    //initalize Sequence object
    buffer = new Sequence(1,getWidth()*getHeight()) ;

    int n = 0;
    ulong acc = 0 ;

    //capturing color from pixel

    // -- bmp file rows are padded to multiples of 4 bytes in 24 bmp files
    int pad = (((bmih.biWidth * 3) % 4) == 0) ? 0 : 4 - ((bmih.biWidth * 3) % 4);

    //    reading pixels from bmp file.
    //    Pay attention that bmp file are BGR ordering. There's a padding at end.
    //    BMP files are stored upside down in the file (180° rotation to display file)
    //    (if height is negative then this way of reading will be incorrect!)

    unsigned char *blue = new unsigned char [1];
    unsigned char *red = new unsigned char [1];
    unsigned char *green = new unsigned char [1];
    unsigned char *gray = new unsigned char [1];

    n = getWidth() * getHeight();

    for(int i = 0;i<bmih.biHeight ;i++){
        for (int j = bmih.biWidth  - 1; j >= 0 ; --j)
        {
            //reading in the values;
            fread(&blue[0], 1, 1, myFile);
            fread(&green[0], 1, 1, myFile);
            fread(&red[0], 1, 1, myFile);

            gray[0] = (float)((((float)(((float)blue[0]*(float)5.0)/(float)32.0))+
                               (float)(((float)green[0]*(float)16.0)/(float)32.0)+
                               (float)(((float)red[0]*(float)11.0)/(float)32.0));

            //filling up the buffer with the grayscaled pixels
            buffer->frames[0][(n-(i*bmih.biWidth)) -j] = (float)((float)(gray[0])/(float)(255.0)) ;

        }
        //reading the row end padding
        unsigned char temp;
        for (int j = 0; j < pad; ++j) {
            fread(&temp, 1, 1, myFile);
        }
    }

    printf("GrayScalling image completed\n") ;

    /* Close stream */
    if( fclose( myFile ) ){
        MessageBox (NULL, _T("file not closed"),
                    _T("ERROR"), MB_OK);

        printf( "The image file to track was not closed\n" );
    }else{
      printf("The image file to track was closed\n") ;

    }//return value
    Sequence *TMPBuffer = buffer ;

    return TMPBuffer ;

}


/*****************************************************************************************************/
/*** Function setWidth(int paramWidth)****************************************************************/
/*****************************************************************************************************/
/*** This function is used to set private member variable                                          ***/
/***                                                                                              ***/
/*** 1 parameters : 1) the width of the image                                                      ***/
/*** return value : -                                                                              ***/
/*****************************************************************************************************/
void WrappedFaceDetector::setWidth(int paramWidth)
{
    _width = paramWidth ;
}

/*****************************************************************************************************/
/*** Function setHeight(int paramHeight)**************************************************************/
/*****************************************************************************************************/
/*** This function is used to set private member variable                                          ***/
/***                                                                                              ***/
/*** 1 parameter  : 1) the height of the image                                                     ***/
/*** return value : -                                                                              ***/
/*****************************************************************************************************/
void WrappedFaceDetector::setHeight(int paramHeight)
{
    _height = paramHeight ;
}

/*****************************************************************************************************/
/*** Function getWidth()******************************************************************************/
/*****************************************************************************************************/
/*** This function is used to get private member variable                                          ***/
/***                                                                                              ***/
/*** parameters   : -                                                                              ***/
/*** return value : the width of the image                                                         ***/
/*****************************************************************************************************/
int WrappedFaceDetector::getWidth()
{
    return _width ;
}

/*****************************************************************************************************/
/*** Function getHeight()*****************************************************************************/
/*****************************************************************************************************/
/*** This function is used to get private member variable                                          ***/
/***                                                                                              ***/
/*** parameters   : -                                                                              ***/
/*** return value : the height of the image                                                        ***/
/*****************************************************************************************************/
int WrappedFaceDetector::getHeight()
{
    return _height ;
}

/*****************************************************************************************************/
/*** Function getSuccess()****************************************************************************/
/*****************************************************************************************************/
/*** This function is used to get private member variable                                          ***/
/***                                                                                              ***/
/*** parameters   : -                                                                              ***/
/*** return value : the success of the face tracking : "0" no face found - "1" face found         ***/
/*****************************************************************************************************/
BOOL WrappedFaceDetector::getSuccess()
{
    return success ;
}
```

Constructeurs :

Comme expliqué plus haut, le constructeur utilisé pour englober les fonctions de « Torch » est le deuxième sur l'image ci-dessous (le premier étant simplement le constructeur par défaut).

Nous retrouvons les 3 paramètres, à savoir :

- le chemin vers le fichier représentant le worldmodel (modèle de visage). Le worldModel utilisé est celui fourni par l'IDIAP et se nomme `FaceDetection.vwm`.
- le chemin vers le fichier .bmp où se trouve l'image avec un visage à détecter (où pas…)
- le type de dégradé de gris (grayscale) à utiliser

Un pointeur vers un objet de type Sequence (de la librairie Torch) est créé dans le constructeur. Cet objet va servir à enregistrer les pixels de l'image. Cependant les pixels doivent être en dégradés de gris.

A cet effet, la fonction loadBitmapFromFile va se charger de faire cela en prenant en paramètre le lien vers le fichier image à traiter et le pointeur vers l'objet de type Sequence. Cette fonction va parcourir le fichier image, faire la conversion des pixels en grayscale (dégradé de gris) et enregistrer les pixels modifier dans le tableau buffer de type Sequence.
Cette fonction retourne un pointeur Sequence vers le tableau de pixels, et c'est l'objet buffer de type Sequence qui va pointer ce tableau de pixels à la sortie de la fonction.

Ensuite le détecteur est initialiser avec la taille de l'image à analyser (hauteur, largeur)
Une fois le détecteur initalisé, il est possible de lancer la fonction `scan` sur le détecteur nouvellement créé (pointeur sur un objet de la classe `FaceDetector` de la librairie Torch)
Si le résultat du scan de l'image par le détecteur retourne une valeurs supérieur à –FLT_MAX(plus grand nombre réel négatif codable sur un float où FLT_MAX=3.402823466e+38F) alors le détecteur a trouvé un visage sur la photo.
En d'autres termes, si la valeur enregistrée dans la variable `success` vaut « 1 », alors le détecteur a trouvé un visage, sinon elle vaudra « 0 » et par voie de conséquence, aucun visage n'a pu être détecté.

Semblable aux fonctions malloc et free en C++, la librairie de l'IDIAP possède ses propres fonctions pour allouer et désallouer de la mémoire afin d'être totalement indépendant des divers systèmes d'exploitations. Afin de désallouer l'espace mémoire qui a été utilisé par les objets créés par le détecteur, on utilise la fonction freeAll() d'allocator.

```
/*** Constructors ********************************************************************************/
WrappedFaceDetector::WrappedFaceDetector()
{
    //default constructor
}

WrappedFaceDetector::WrappedFaceDetector(char *worldModelPath,char *fileName,char grayMode)
: _pDetector(NULL), _newFaceAvailable(false) , _movementSensitivity(DefaultMovementSensitivity)
{

    //  Check if the WorldModelFile exists
/*  if(fopen(worldModelPath,"r")==0)
    {
        MessageBox (NULL, _T("ERROR:WorldModelFile doesn't exist!%c",worldModelPath),
                    _T("ERROR"), MB_OK);

    }
    else
    {*/
        //buffer is a Torch Sequence object that stores the grayscale pixel from the image
        //for the face tracking's algorithm
        Sequence *buffer;

        //file is loaded, grayscaled and stored in buffer object
        buffer = loadBitmapFromFile(fileName, buffer) ;


        //initialize detector
        _pDetector = new Torch::FaceDetector((char *)worldModelPath , getWidth() , getHeight()) ;

        //scan buffer & hope we find a face... ;)
        success = (_pDetector->scan(buffer)>-FLT_MAX);

        //_pDetector->~FaceDetector();


        _pDetector->allocator->freeAll();


    /*}*/

}
```

buffer est un tableau remplis de pixels en dégradé de gris de l'image à analyser

Création du détecteur de visage et scan du tableau de pixels à la recherche d'un visage

Destructeur :

Afin de libérer la mémoire, il faut détruire le pointeur qui pointe vers le détecteur quand on utilise le destructeur de la classe enveloppe (WrappedFaceDetector).

```
/*** Destructor ********************************************************************************/
WrappedFaceDetector::~WrappedFaceDetector()
{
    //Free memory
    if(_pDetector)
    {
        delete _pDetector;

    }
}
```

S'il y a un détecteur qui a été créé, ce dernier sera détruit pour libérer la mémoire

Fonction : LoadBitmapFromFile

Cette fonction prend deux paramètres :

- le fichier image à analyser dans le but de lire les pixels
- le pointeur vers l'objet buffer de type Sequence(Torch) dans le but d'enregistrer les pixels en dégradé de gris

Cette fonction retourne un pointeur vers un objet de type Sequence(Torch) avec le tableau de pixels en dégradé de gris de l'image à analyser

```
/*****************************************************************************************************/
/*** Function loadBitmapFromFile(char *fileName, Sequence *buffer)*********************************/
/*****************************************************************************************************/
/*** This function is used to grayscale an bmp image in order to prepare Torch's Sequence       ***/
/*** object for face tracking                                                                    ***/
/***                                                                                            ***/
/*** 2 parameters : 1) the image to track the face                                               ***/
/***                2) the buffer to store the graycaled pixel                                  ***/
/*** return value : pointer to a Sequence object (Torch Library object                           ***/
/*****************************************************************************************************/
Sequence* WrappedFaceDetector::loadBitmapFromFile(char *fileName, Sequence *buffer)
{
    //Local Declarations
    char *pStr1 = fileName;

    FILE *myFile ;
    //Bitmap sections
    BITMAPFILEHEADER bmfh ;
    BITMAPINFOHEADER bmih ;

    //Opening file
    if((myFile = fopen(pStr1, "rb"))==NULL){
            MessageBox (NULL, _T("file not opened"),
                        _T("ERROR"), MB_OK);
        printf( "The file with the face to be tracked was not opened\n" ) ;
    }else{

            printf ("The file with the face to be tracked is opened\n") ;}
    //check if file is opened...
    if(myFile==NULL)
    {
        MessageBox (NULL, _T("ERROR:File not opened!"),
                    _T("ERROR"), MB_OK);
    }

    //Reading information header & file header
    fread(&bmfh , sizeof(BITMAPFILEHEADER) , 1 , myFile) ;
    fread(&bmih , sizeof(BITMAPINFOHEADER) , 1 , myFile) ;

    printf("Bits per pixel = %d\n", bmih.biBitCount) ;
    //Is the BMPFile 24 bits ? (To test with device in release mode)
    /*if(bmih.biBitCount == 24)
    {
        MessageBox (NULL, _T("BMPFile : 24 Bits"),
                    _T("INFORMATION"), MB_OK);
    }
    else
    {
        MessageBox (NULL, _T("BMPFile : not a 24 Bits"),
                    _T("INFORMATION"), MB_OK);
    }
    */



    //set width& height
    setHeight(bmih.biHeight) ;
    setWidth(bmih.biWidth)   ;

    //initalize Sequence object
    buffer = new Sequence(1,getWidth()*getHeight()) ;

    int n = 0;
    ulong acc = 0 ;

    //capturing color from pixel

    // -- bmp file rows are padded to multiples of 4 bytes in 24 bmp files
    int pad = (((bmih.biWidth * 3) % 4) == 0) ? 0 : 4 - ((bmih.biWidth * 3) % 4);

    //    reading pixels from bmp file.
    //    Pay attention that bmp file are BGR ordering. There's a padding at end.
    //    BMP files are stored upside down in the file (180° rotation to display file)
    //    (if height is negative then this way of reading will be incorrect!)

    unsigned char *blue = new unsigned char [1];
    unsigned char *red = new unsigned char [1];
    unsigned char *green = new unsigned char [1];
    unsigned char *gray = new unsigned char [1];

    n = getWidth() * getHeight();

    for(int i = 0;i<bmih.biHeight ;i++){
        for (int j = bmih.biWidth  - 1; j >= 0 ; --j)
        {
            //reading in the values;
            fread(&blue[0], 1, 1, myFile);
            fread(&green[0], 1, 1, myFile);
            fread(&red[0], 1, 1, myFile);

            gray[0] = (float)(((float)(((float)blue[0]*(float)5.0)/(float)32.0))+
                              (float)(((float)green[0]*(float)16.0)/(float)32.0)+
                              (float)(((float)red[0]*(float)11.0)/(float)32.0));

            //filling up the buffer with the grayscaled pixels
            buffer->frames[0][(n-(i*bmih.biWidth)) -j] = (float)((float)(gray[0])/(float)(255.0)) ;

        }
        //reading the row end padding
        unsigned char temp;
        for (int j = 0; j < pad; ++j) {
            fread(&temp, 1, 1, myFile);
        }
    }

    printf("GrayScalling image completed\n") ;

    /* Close stream */
    if( fclose( myFile ) ){
        MessageBox (NULL, _T("file not closed"),
                    _T("ERROR"), MB_OK);

        printf( "The image file to track was not closed\n" );
    }else{
      printf("The image file to track was closed\n") ;

    }//return value
    Sequence *TMPBuffer = buffer ;

    return TMPBuffer ;

}
```

Informations générales sur le fichier image bmp :
- headers du fichier
- taille de l'image

Le tableau de « Sequence » doit être créé avec la taille de l'image à analyser

L'image en « grayscale » (dégradé de gris) doit être enregistré dans le tableau de « Sequence »

Au sortir de la fonction, nous remarquons que le pointeur vers le tableau de type Sequence est retourné.

Mutateurs

Fonction : setWidth

Cette fonction prend un paramètre :

- un entier qui permet d'affecter une valeur au membre privé `_width`

```
/*****************************************************************************************************/
/*** Function setWidth(int paramWidth)*****************************************************************/
/*****************************************************************************************************/
/*** This function is used to set private member variable                                          ***/
/***                                                                                               ***/
/*** 1 parameters : 1) the width of the image                                                      ***/
/*** return value : -                                                                              ***/
/*****************************************************************************************************/
void WrappedFaceDetector::setWidth(int paramWidth)
{
    _width = paramWidth ;
}
```

Fonction : setHeight

Cette fonction prend un paramètre :

- un entier qui permet d'affecter une valeur au membre privé `_height`

```
/*****************************************************************************************************/
/*** Function setHeight(int paramHeight)***************************************************************/
/*****************************************************************************************************/
/*** This function is used to set private member variable                                          ***/
/***                                                                                               ***/
/*** 1 parameter  : 1) the height of the image                                                     ***/
/*** return value : -                                                                              ***/
/*****************************************************************************************************/
void WrappedFaceDetector::setHeight(int paramHeight)
{
    _height = paramHeight ;
}
```

Accesseurs

Fonction : getWidth

Cette fonction retourne une valeur :

- un entier qui permet de récupérer la valeur du membre privé `_width`

```
/*****************************************************************************************************/
/*** Function getWidth()*******************************************************************************/
/*****************************************************************************************************/
/*** This function is used to get private member variable                                          ***/
/***                                                                                               ***/
/*** parameters   : -                                                                              ***/
/*** return value : the width of the image                                                         ***/
/*****************************************************************************************************/
int WrappedFaceDetector::getWidth()
{
    return _width ;
}
```

Fonction : getHeight

Cette fonction retourne une valeur :

- un entier qui permet de récupérer la valeur du membre privé `_height`

```
/*****************************************************************************************************/
/*** Function getHeight()*****************************************************************************/
/*****************************************************************************************************/
/*** This function is used to get private member variable                                          ***/
/***                                                                                              ***/
/*** parameters    : -                                                                             ***/
/*** return value : the height of the image                                                       ***/
/*****************************************************************************************************/
int WrappedFaceDetector::getHeight()
{
    return _height ;
}
```

Fonction : getSuccess

Cette fonction retourne une valeur :

- un booléen qui permet de récupérer la valeur du membre privé `success` qui vaut « 0 » si aucun visage n'a été détecté et « 1 » si un visage à été détecté sur l'image

```
/*****************************************************************************************************/
/*** Function getSuccess()****************************************************************************/
/*****************************************************************************************************/
/*** This function is used to get private member variable                                          ***/
/***                                                                                              ***/
/*** parameters    : -                                                                             ***/
/*** return value : the success of the face tracking : "0" no face found - "1" face found         ***/
/*****************************************************************************************************/
BOOL WrappedFaceDetector::getSuccess()
{
    return success ;
}
```

Explication supplémentaire sur la fonction LoadBitmapFromFile et la manière dont les pixels sont lus pour détecter un visage :

Dans un fichier BMP, les pixels de l'image ne sont pas stockés tels que nous les voyons. En effet l'image vue à l'écran est stockée de façon inversée dans le fichier. Il y a une symétrie d'axe horizontale (flip vertical).
En effet prenons l'exemple ci-dessous, où nous distinguons clairement les 4 régions de l'image mises en évidences par les couleurs suivantes :
rouge – vert – bleu – jaune.

L'image telle que vue à l'écran est la suivante :

Cependant cette image vue à l'écran est stockée de la façon suivante dans le fichier .bmp :

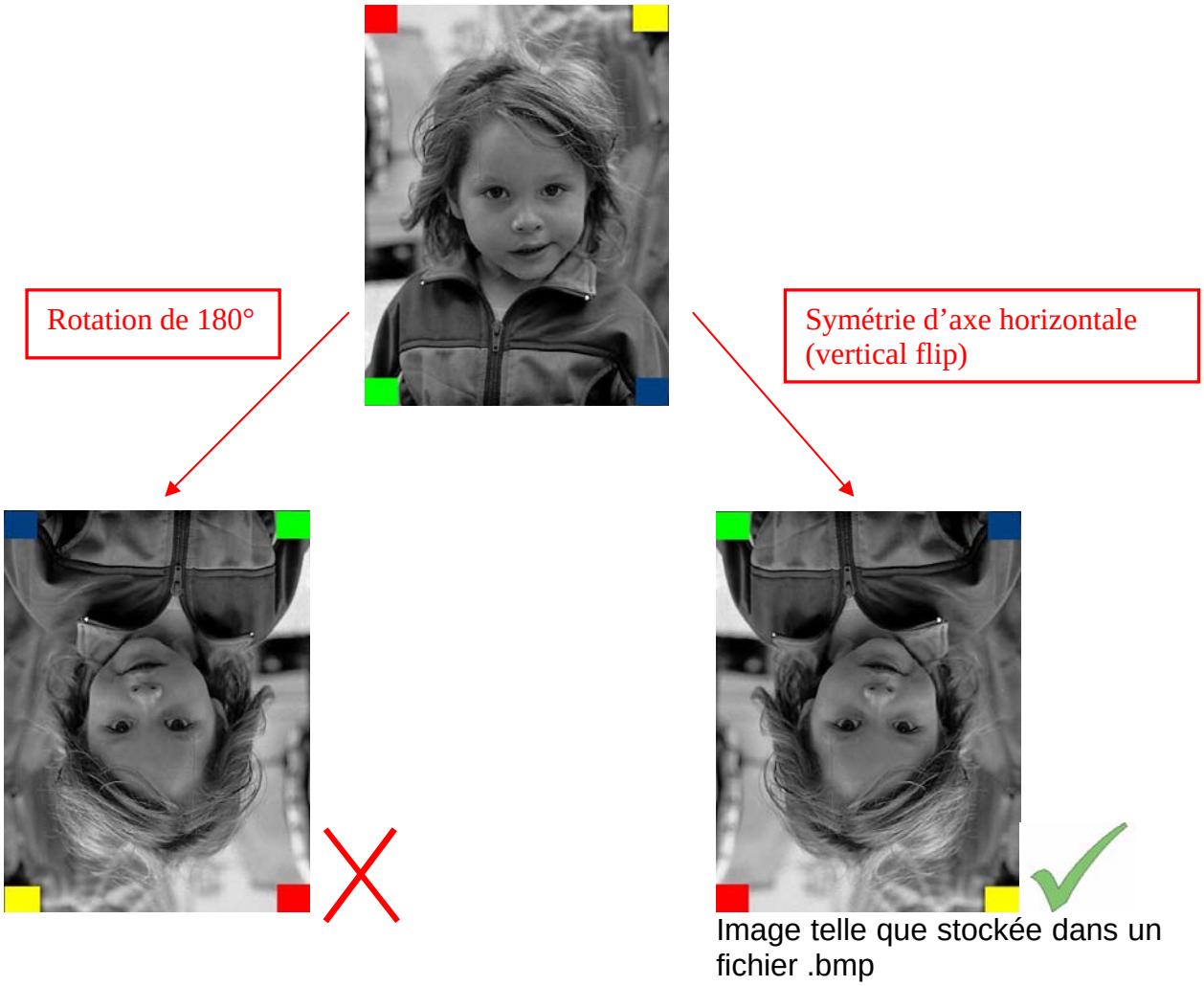
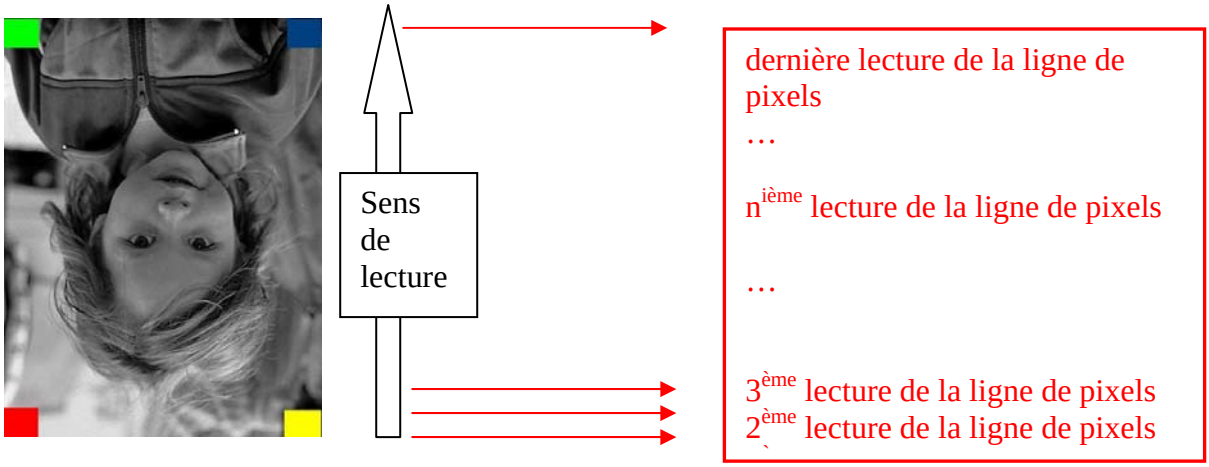


Image telle que stockée dans un fichier .bmp

Il faut faire attention de ne pas confondre une rotation de 180° avec une symétrie d'axe horizontale (telle que se trouve stockée l'image dans un fichier .bmp).

A cet effet, nous remarquons que dans tout les cas il faut parcourir le fichier depuis le côté rouge (début de l'image) jusqu'au côté bleu (fin de l'image) afin d'être cohérent avec l'image original.

Il est à dans cette optique aisé de comprendre que le fichier doit être parcouru depuis le côté en bas à gauche (en rouge sur l'image ci-dessous) pour finir sur le côté en haut à droite (en bleu sur l'image ci-dessous).



Etant donné que nous sommes sur un codage RGB 24 bits, alors cela signifie que chaque pixel est codé sur 3 bytes (3 bytes * 8 = 24 bits). Etant donné que chaque pixel est composé des couleurs R(Red – rouge), G(Green – vert), B(Blue – bleu), cela signifie que chacune des couleurs est codée sur un byte.
Etant donné que nous lisons le fichier dans le sens opposé, il faut faire attention au fait que l'ordre dans lequel nous lisons n'est plus RGB mais BGR.

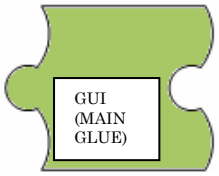
Dans le code qui suit nous remarquons l'utilisation de la fonction `fread` de `stdlib.h` qui par défaut lit un byte à la fois. Cependant, étant donné que fread lit dans le sens normal du fichier (de haut en bas), il nous faut enregistrer dans le tableau buffer de type Sequence depuis la dernière ligne jusqu'à la première :

```
    for(int i = 0;i<bmih.biHeight ;i++){
        for (int j = bmih.biWidth  - 1; j >= 0 ; --j)
        {
            //reading in the values;
            fread(&blue[0], 1, 1, myFile);
            fread(&green[0], 1, 1, myFile);
            fread(&red[0], 1, 1, myFile);

            gray[0] = (float)((((float)(((float)blue[0]*(float)5.0)/(float)32.0))+
                               (float)(((float)green[0]*(float)16.0)/(float)32.0)+
                               (float)(((float)red[0]*(float)11.0)/(float)32.0));

            //filling up the buffer with the grayscaled pixels
            buffer->frames[0][(n-(i*bmih.biWidth)) -j] = (float)((float)(gray[0])/(float)(255.0)) ;

        }
        //reading the row end padding
        unsigned char temp;
        for (int j = 0; j < pad; ++j) {
            fread(&temp, 1, 1, myFile);
        }
}

    printf("GrayScalling image completed\n") ;
```

Compilation et link

Cette ancienne librairie prend entre 10 et 15 secondes pour trouver un visage. Nous accusons comme tout pour la nouvelle libraire un problème avec le manque de FPU dans le PDA.

Le GUI maître de l'application



GUI (GRAPHICAL USER INTERFACE

Nous allons à présent voir l'implémentation de notre dernier, et non des moindres, bloc de notre jeu. En effet celui-ci constitue la « pièce maîtresse » qui complète le puzzle de notre application.
Ses seuls et uniques objectifs sont ceux de:

- faire appel aux DLL et de lancer les processus (ou threads) que nous avions souhaités à savoir :
  a)un « thread » permettant de faire l'abstraction de DirectShow. Il ne fera que deux choses principalement :
  - fournir la possibilité d'avoir une « preview video window » du stream vidéo provenant de la caméra
  - prendre une image de ce même flux vidéo et de le stocker dans un fichier .bmp.

  b) un « thread » permettant de faire l'abstraction du code de la libraire de l'IDIAP en ne fournissant qu'une seule chose :
  - la possibilité de détecter un visage sur une image que l'on aurait passé en entrée.
- d'offrir principalement une fonctionnalité de dessin sur le flux vidéo. Les objets à dessiner sont :
  a) un rectangle spécifiant le visage sur une image
  b) deux rectangles spécifiant les coordonnées des yeux

Ci-dessous, nous pouvons voir la *solution* 'FT_FaceTracker' avec un projet C# « FT_GUI ». C'est ce projet qui englobe tout les fichiers nous permettant de démarrer complètement l'application en lançant les « threads ». Il comprend également les outils pour dessiner.

Figure 74 : Solution Explorer

En ouvrant ce projet, nous remarquons un certain nombre de fichiers avec des classes qui ont été implémentées.

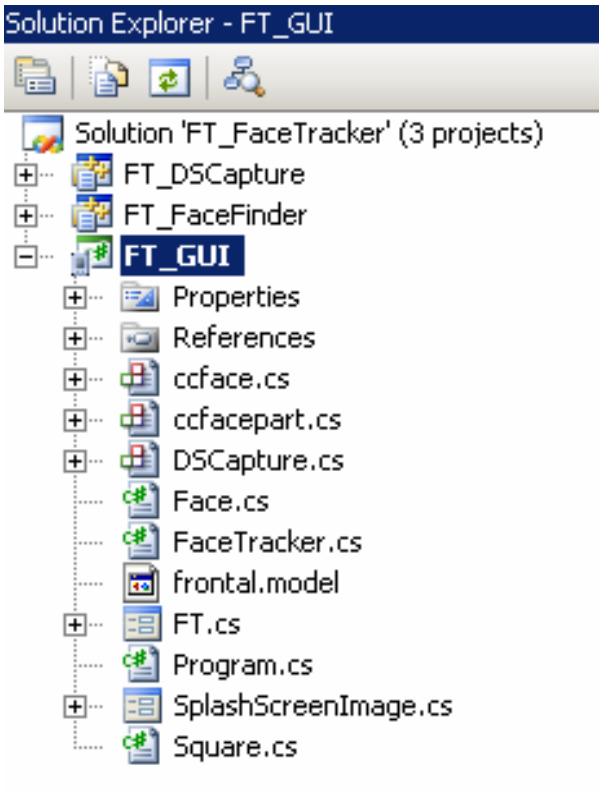


Figure 75 : FT_GUI solution

Où nous retrouvons notamment :

- 2 Windows Form (Designer avec leur code implémenté dans un fichier .cs bien sûr)
- 4 fichier .cs individuels
- 3 Component Class
- 1 fichier de donnée (« content ») qui se trouve être le model de détection de visage : frontal.model

Ci-dessous nous retrouvons les divers « items » qu'il est possible d'avoir dans un projet C#.

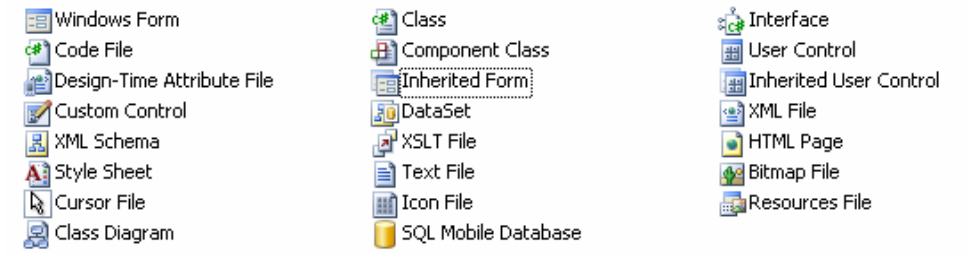


Figure 76 : items dans un projet C#

Ce projet comprend :

- le fichier `Program.cs` possède le point d'entrée de l'application avec la fonction `main()`.
- le windows form `SplashScreenImage.cs` avec son designer et son fichier source du même nom. Cette classe est uniquement utilisé pour lancer le « logo de l'application » au démarrage.
- le windows form `FT.cs` avec son designer et son fichier source du même nom. Cette classe est LA classe importante du projet, en ce sens où, elle dispose de l'interface graphique avec laquelle l'utilisateur va interagir. Tout se passe dans cette classe. D'ailleurs, la fonction main() de la classe `Program.cs` va lancer une instance de cette classe au démarrage son application.
- le Component Class `DSCapture.cs` qui va faire les appels aux fonctions importées depuis la DLL « FT_DSCapture.dll ». Elle va permettre de regrouper notamment toute la partie abstraite de DirectShow dans un simple objet
- le Component Class `ccfacepart.cs` qui met en place l'outillage de dessin
- la classe `Square.cs` permet de dessiner et mettre à jour un carré sur la zone graphique spécifique en indiquant le chemin d'accès à cette zone. Il faudra de plus fournir les informations relatives au dessin souhaité. Cette classe va notamment faire appel à l'outil de dessin fourni par la classe `ccfacepart.cs` dans le but de créer une forme carrée vide à l'intérieur.
- la classe `Face.cs` permet de créer un model de visage à dessiner, en spécifiant notamment qu'un visage comporte 3 carrés, à savoir, celui mettant en relief le visage et deux autres indiquant les yeux.
- la classe `FaceTracker.cs` permet de faire appel à la fonction track(…) exportée depuis la DLL « FT_FaceFinder.dll », en lui passant tout les paramètres nécessaires.

Explications détaillés du code se trouvant dans les classes :

Une brève explication sur certains fichiers est founis en annexe, mais pour plus de précisions, prière de vous référer au code développés..

A retenir de tout ceci : un schéma

Le sujet est vaste est tellement nouveau qu'il y aurait encore beaucoup à dire, mais si nous devons retenir quelque chose de cette application, c'est bien la manière dont fonctionne le programme actuellement : (afin peut-être d'y apporter de futurs modifications)

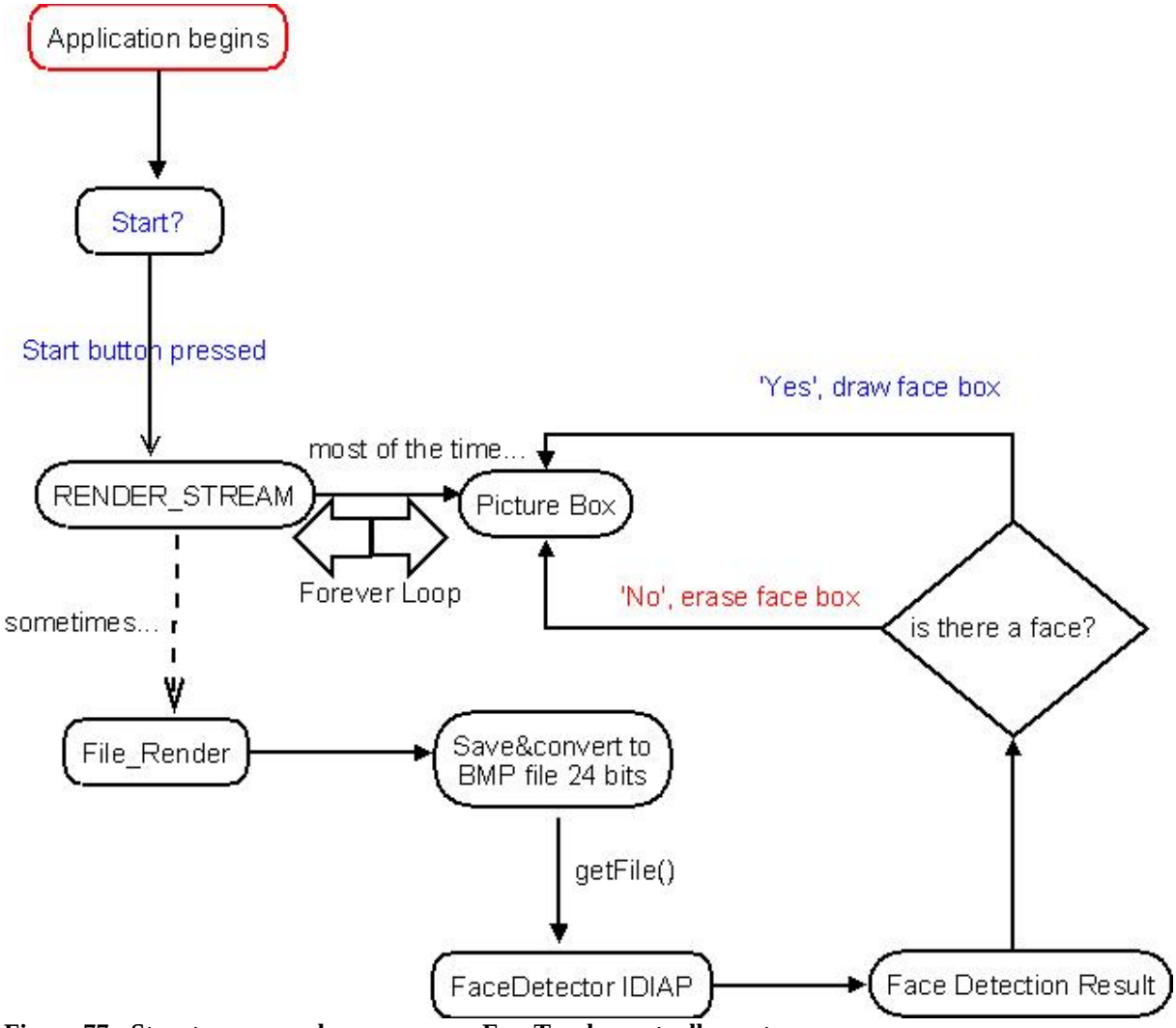


Figure 77 : Structogramme du programme FaceTracker actuellement

Tests et problèmes rencontrés

Certains PDA montraient une impossibilité de rendre un stream vidéo :
Notamment pour le HTC Touch qui possèdent apparemment ses propres filtres qui rentrent en conflit avec les filtres DirectShow développés dans le cadre du projet.

Le SPV M310 montrait des problèmes de symétrie de l'image.

Le PDA HTC P3300 permet quant à lui de parfaitement faire tourner l'application de démonstration du « Face Tracker »

Une série de programme de tests a pu être développé (malheureusement sans beacoup de commentaires) mais qui permettent de tester des choses tels que (dessiner sur un flux vidéo, faire le grayscalling d'une image…)

Ces projets se trouvent « en vrac » sur le CD.

Le projet de diplôme « FaceTracker », les démonstrateurs pour FaciaMea(MemoriaMea) et les divers projets de tests sont disponibles sur les CD.

Remarque : une série de logos différents pour l'application ont été créé également.

Conclusion préalables : Améliorations possibles

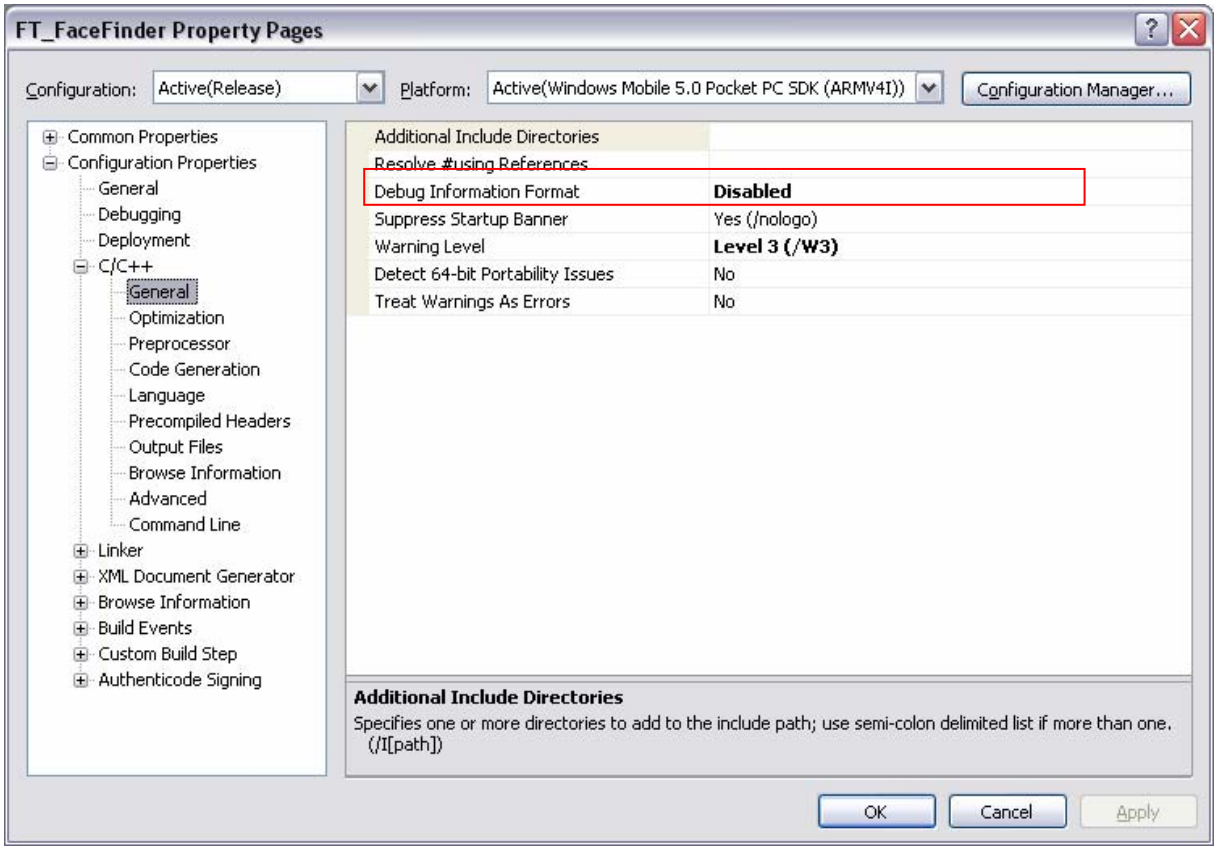
Il pourrait y avoir plusieurs chemins permettant de déterminer les futurs ou probables chemins amenant à une rapidité de la détection de visage :

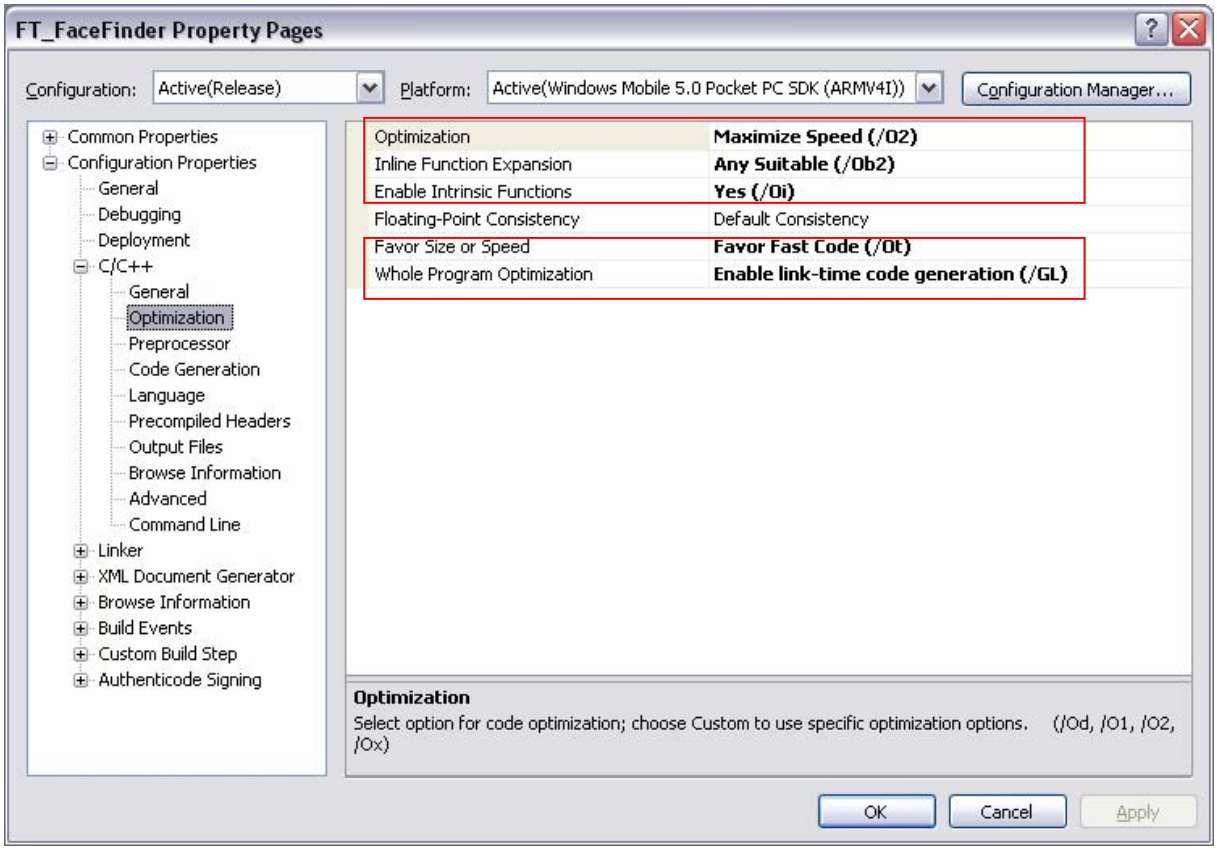
- passer un tableau de byte plutôt qu'un fichier .bmp en entrée à l'algorithme
- trouver un moyen de transmettre un « raw image » depuis DirectShow plutôt que de perdre du temps à faire tout le processus de sauvegarde
- implémenter un filtre performant personnel de « grabbing image » dans DirectShow
- Trouver les valeurs optimales pour les paramètres des fonctions de l'API de l'IDIAP.

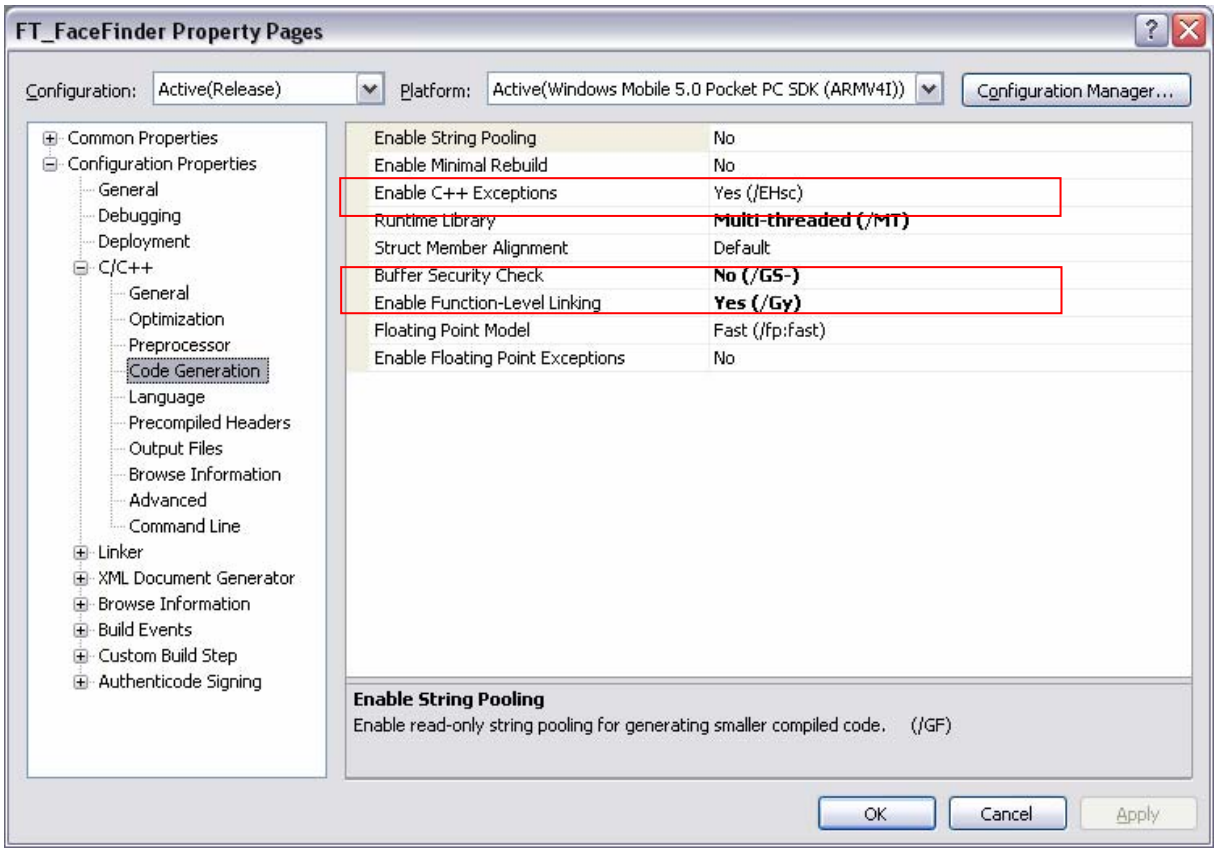
Optimisations venant du projet VisualStudio 2005

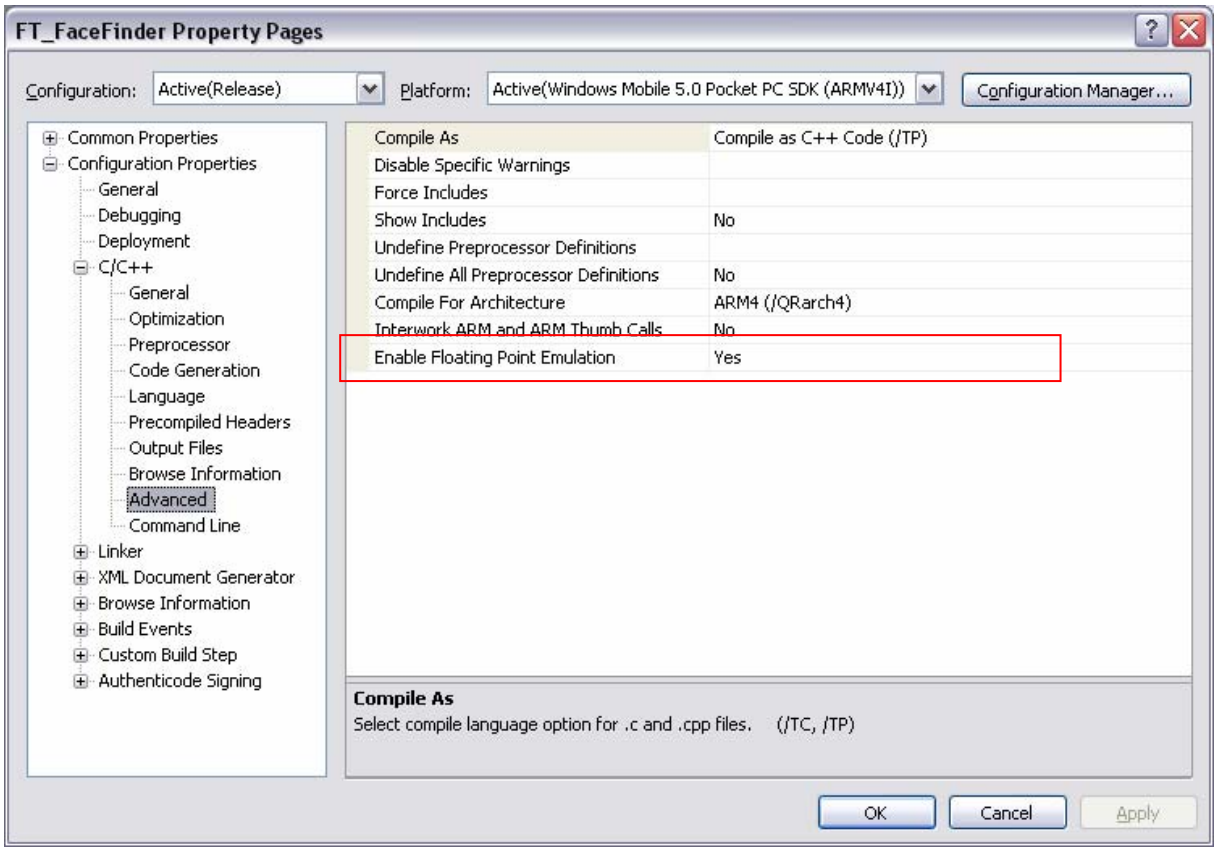
Il est toutefois bon de noter que des optimisations dans la compilation et le link des projets sont possibles de réaliser :

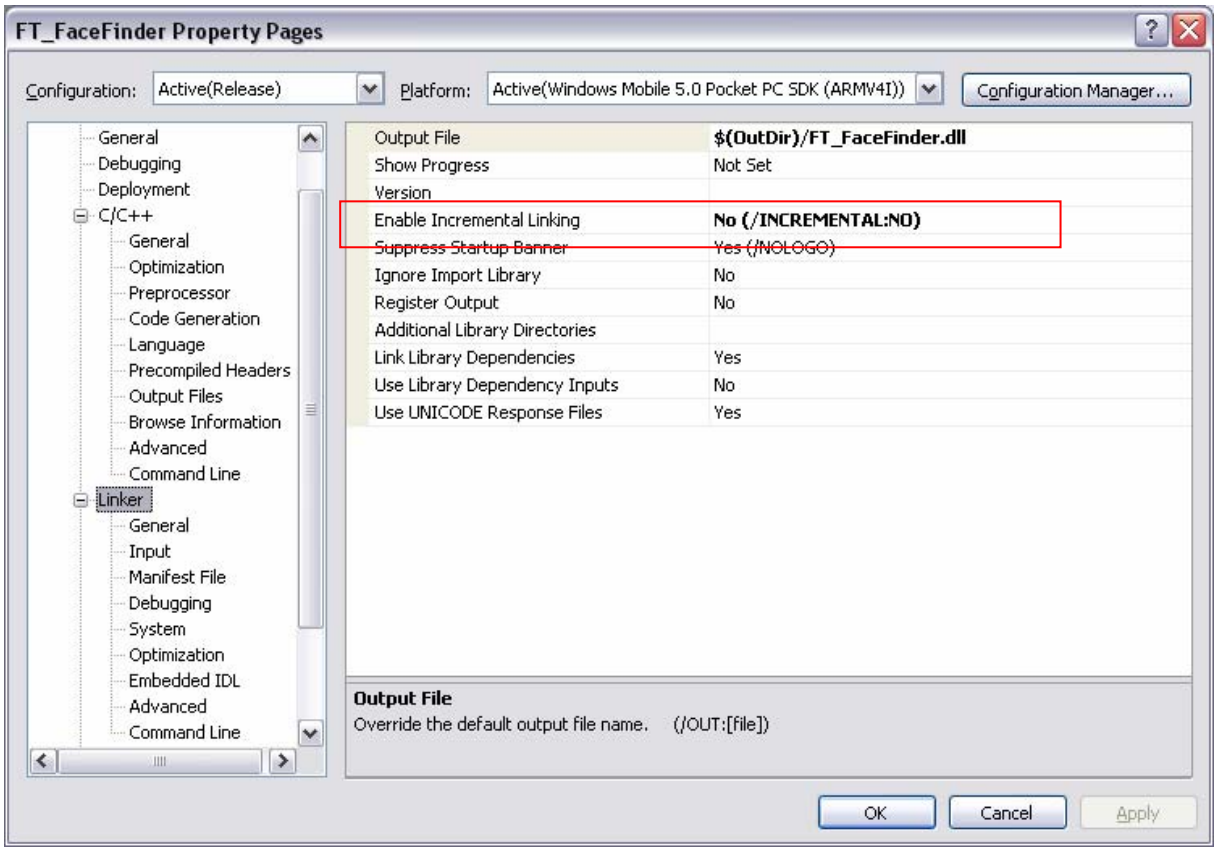
Voici indiqué en rouge les paramètres à changer dans les projets :
(Notamment pour les DLL englobant le code DirectShow et le code de détection de l'IDIAP) :

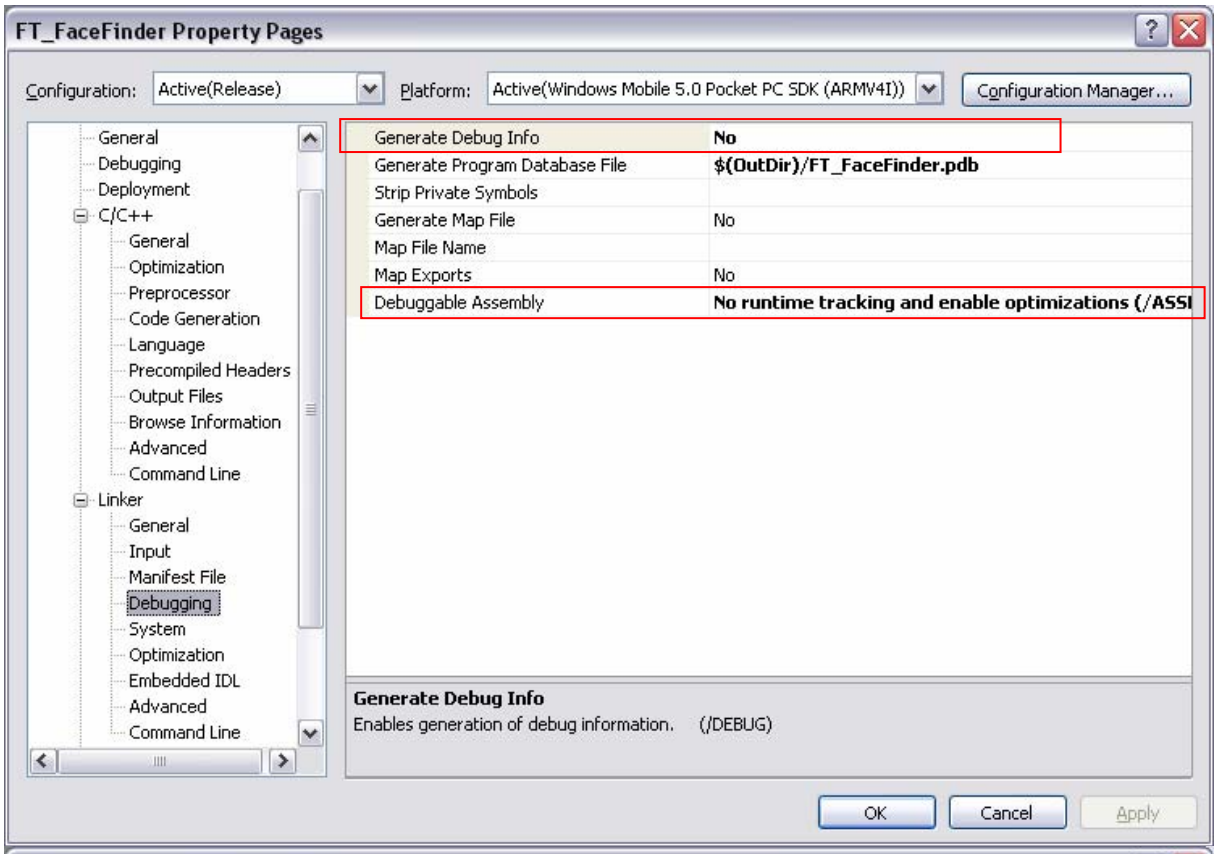

FT_FaceFinder Property Pages
Configuration: Active(Release)
Platform: Active(Windows Mobile 5.0 Pocket PC SDK (ARMV4I))
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
Deployment
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
Advanced
Command Line
Linker
XML Document Generator
Browse Information
Build Events
Custom Build Step
Authenticode Signing
Additional Include Directories
Resolve #using References
Debug Information Format
Disabled
Suppress Startup Banner
Yes (/nologo)
Warning Level
Level 3 (/W3)
Detect 64-bit Portability Issues
No
Treat Warnings As Errors
No
Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])
OK
Cancel
Apply

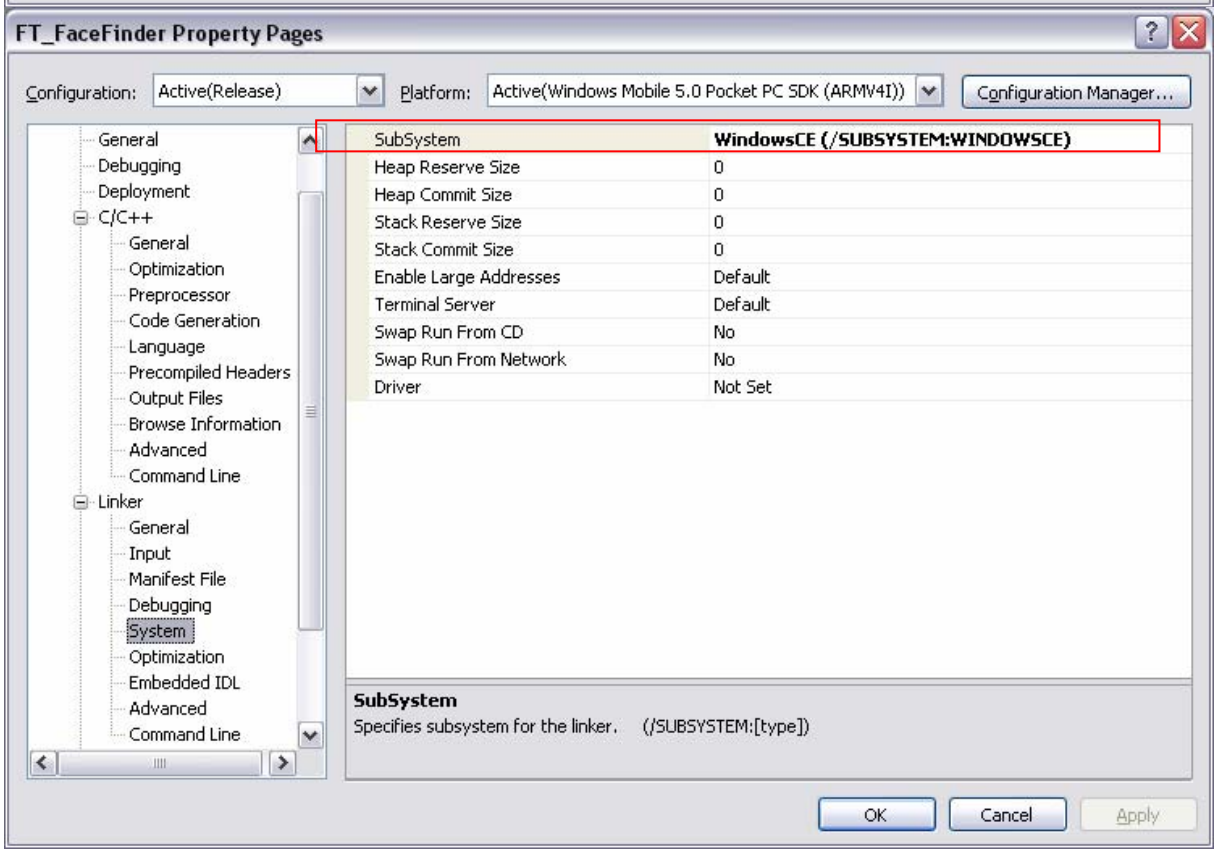

FT_FaceFinder Property Pages
Configuration: Active(Release)
Platform: Active(Windows Mobile 5.0 Pocket PC SDK (ARMV4I))
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
Deployment
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
Advanced
Command Line
Linker
XML Document Generator
Browse Information
Build Events
Custom Build Step
Authenticode Signing
Optimization | Maximize Speed (/O2)
Inline Function Expansion | Any Suitable (/Ob2)
Enable Intrinsic Functions | Yes (/Oi)
Floating-Point Consistency | Default Consistency
Favor Size or Speed | Favor Fast Code (/Ot)
Whole Program Optimization | Enable link-time code generation (/GL)
Optimization
Select option for code optimization; choose Custom to use specific optimization options. (/Od, /O1, /O2, /Ox)
OK
Cancel
Apply

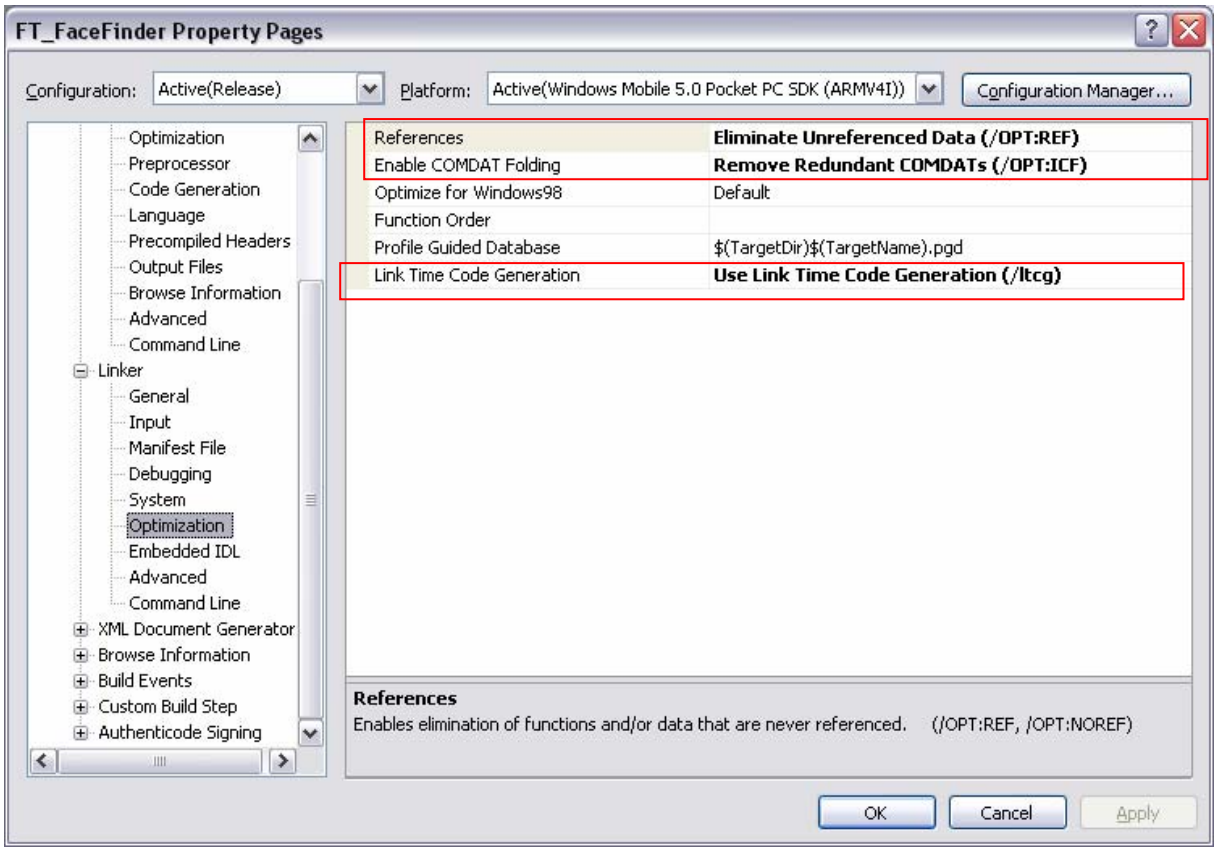

FT_FaceFinder Property Pages
Configuration: Active(Release)
Platform: Active(Windows Mobile 5.0 Pocket PC SDK (ARMV4I))
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
Deployment
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
Advanced
Command Line
Linker
XML Document Generator
Browse Information
Build Events
Custom Build Step
Authenticode Signing
Enable String Pooling | No
Enable Minimal Rebuild | No
Enable C++ Exceptions | Yes (/EHsc)
Runtime Library | Multi-threaded (/MT)
Struct Member Alignment | Default
Buffer Security Check | No (/GS-)
Enable Function-Level Linking | Yes (/Gy)
Floating Point Model | Fast (/fp:fast)
Enable Floating Point Exceptions | No
Enable String Pooling
Enable read-only string pooling for generating smaller compiled code. (/GF)
OK
Cancel
Apply


FT_FaceFinder Property Pages
Configuration: Active(Release)
Platform: Active(Windows Mobile 5.0 Pocket PC SDK (ARMV4I))
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
Deployment
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
Advanced
Command Line
Linker
XML Document Generator
Browse Information
Build Events
Custom Build Step
Authenticode Signing
Compile As
Compile as C++ Code (/TP)
Disable Specific Warnings
Force Includes
Show Includes
No
Undefine Preprocessor Definitions
Undefine All Preprocessor Definitions
No
Compile For Architecture
ARM4 (/QRarch4)
Interwork ARM and ARM Thumb Calls
No
Enable Floating Point Emulation
Yes
Compile As
Select compile language option for .c and .cpp files. (/TC, /TP)
OK
Cancel
Apply


FT_FaceFinder Property Pages
Configuration: Active(Release)
Platform: Active(Windows Mobile 5.0 Pocket PC SDK (ARMV4I))
Configuration Manager...
General
Debugging
Deployment
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
Advanced
Command Line
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Advanced
Command Line
Output File
$(OutDir)/FT_FaceFinder.dll
Show Progress
Not Set
Version
Enable Incremental Linking
No (/INCREMENTAL:NO)
Suppress Startup Banner
Yes (/NOLOGO)
Ignore Import Library
No
Register Output
No
Additional Library Directories
Link Library Dependencies
Yes
Use Library Dependency Inputs
No
Use UNICODE Response Files
Yes
Output File
Override the default output file name. (/OUT:[file])
OK
Cancel
Apply


FT_FaceFinder Property Pages
Configuration: Active(Release)
Platform: Active(Windows Mobile 5.0 Pocket PC SDK (ARMV4I))
Configuration Manager...
General
Debugging
Deployment
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
Advanced
Command Line
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Advanced
Command Line
Generate Debug Info No
Generate Program Database File $(OutDir)/FT_FaceFinder.pdb
Strip Private Symbols
Generate Map File No
Map File Name
Map Exports No
Debuggable Assembly No runtime tracking and enable optimizations (/ASSI
Generate Debug Info
Enables generation of debug information. (/DEBUG)
OK
Cancel
Apply


FT_FaceFinder Property Pages
Configuration: Active(Release)
Platform: Active(Windows Mobile 5.0 Pocket PC SDK (ARMV4I))
Configuration Manager...
General
Debugging
Deployment
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
Advanced
Command Line
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Advanced
Command Line
SubSystem WindowsCE (/SUBSYSTEM:WINDOWSCE)
Heap Reserve Size 0
Heap Commit Size 0
Stack Reserve Size 0
Stack Commit Size 0
Enable Large Addresses Default
Terminal Server Default
Swap Run From CD No
Swap Run From Network No
Driver Not Set
SubSystem
Specifies subsystem for the linker. (/SUBSYSTEM:[type])
OK
Cancel
Apply


FT_FaceFinder Property Pages
Configuration: Active(Release)
Platform: Active(Windows Mobile 5.0 Pocket PC SDK (ARMV4I))
Configuration Manager...
Optimization
Preprocessor
Code Generation
Language
Precompiled Headers
Output Files
Browse Information
Advanced
Command Line
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Advanced
Command Line
XML Document Generator
Browse Information
Build Events
Custom Build Step
Authenticode Signing
References
Eliminate Unreferenced Data (/OPT:REF)
Enable COMDAT Folding
Remove Redundant COMDATs (/OPT:ICF)
Optimize for Windows98
Default
Function Order
Profile Guided Database
$(TargetDir)$(TargetName).pgd
Link Time Code Generation
Use Link Time Code Generation (/ltcg)
References
Enables elimination of functions and/or data that are never referenced. (/OPT:REF, /OPT:NOREF)
OK
Cancel
Apply

Conclusion

L'application « Face Tracker » qui a été développée peut être considérée comme un véritable *démonstrateur* permettant de dire que la voie est ouverte pour se lancer dans l'aventure de la réalisation d'applications multimodales en général pour les téléphones mobiles Windows Mobile.

En effet, dans le domaine de l'image nous avons pu nous rendre compte que l'application de détection et de suivi de visage a pu être complètement implémentée en nous basant :

- sur l'API DirectShow : pour l'acquisition, le traitement et le rendu d'un flux de donnée vidéo en provenance de la caméra du PDA
- sur les librairies de l'IDIAP (Torch3Vision et FaceFindingAPI_0.2.2 : pour la détection de visage dans une image.
- sur la possibilité de dessiner rapidement sur un flux vidéo avec le langage C# dans le contexte du .NET FrameWork.

La communauté DirectShow pour Windows Mobiles est malheureusement à l'heure actuelle (novembre 2007) encore très peu développée ce qui peut encore être un inconvénient pour débuter rapidement dans le domaine. Cependant ce rapport et le code qui a été développé sur la partie DirectShow sur Windows Mobile devraient être désormais suffisamment explicites pour permettre à tout un chacun de démarrer très rapidement dans le « monde » DirectShow en intégrant rapidement les divers concepts qui s'y rattache.

Quant à la nouvelle librairie de l'IDIAP, elle est très intéressante du point de vue des paramètres exposés par rapport à l'ancienne, en effet, l'utilisateur a plus de possibilité de pouvoir interagir avec les performances ou la qualité de la détection de visage, par le jeu de certains de ces paramètres. De plus une documentation claire et des exemples concrets permettent de rapidement intégrer les nouveaux concepts nécessaires à la compréhension de l'API.

A la conclusion de ce rapport, un autre projet avec la version émulateur de Windows Mobile est en cours. A savoir, démarrer un petit film depuis un fichier vidéo se trouvant sur l'émulateur de PDA (Windows Mobile 5.0 ou 6.0) et de faire une détection de visage sur ce petit film.
Malheureusement le projet étant en cours en parallèle de la rédaction de ce rapport, il ne figurera pas dedans…

Signature

Flavio TARSETTI

Sources

- Kim Roose (assistant HES-SO Valais en Infotronics) : connaissance du .NET FrameWork, aide dans la philosophie du code du projet
- Le livre OPC Fundamentals, Implementation and Application de Frank Iwanitz&Jürgen Lange : explications des objets COM et leurs initializations
- Google.ch pour diverses recherchers
- MSDN : informations concernant Windows Mobile ou DirectShow
- Wikipedia.com : recherché : DirectShow, DirectX
-

Liens utiles

--PropertyBag
---

http://www.gikuzone.org/ebook/Developer/Inside_COM+Base_Services/210.htm

http://blogs.msdn.com/medmedia/

http://blogs.msdn.com/medmedia/archive/2007/01/16/multichannel-audio-in-windows-ce.aspx

http://www.eggheadcafe.com/software/aspnet/29140905/directshow-mobile50-bad.aspx

---
--Writing DirectShow Filters
---
http://msdn2.microsoft.com/en-us/library/ms925289.aspx

http://msdn2.microsoft.com/en-us/library/ms925297.aspx

--Liste des encodeurs vidéos
---
http://forum.hardware.fr/hfr/VideoSon/Traitement-Video/quel-compression-direct-sujet_94028_1.htm

---
--DMO (MSDN)
---
http://msdn2.microsoft.com/en-us/library/ms783356.aspx

--Examples DirectShow on XP
---

http://www.codeguru.com/cpp/g-m/directx/directshow/

//Simultaneous Previewing & Video Capture using DirectShow
http://www.codeguru.com/cpp/g-m/directx/directshow/article.php/c6973/

//DirectShow Single-Frame Capture Class Without MFC
http://www.codeguru.com/cpp/g-m/directx/directshow/article.php/c9551/

--Working with Pictures, Video, and Cameras in Windows Mobile 5.0 (msdn)
---
http://msdn2.microsoft.com/en-us/library/aa454909.aspx#working_with_multimedia_topic7

-DirectShow
///good link (msdn)
---
http://msdn2.microsoft.com/en-us/library/ms923367.aspx
http://msdn2.microsoft.com/en-us/library/aa454909.aspx#working_with_multimedia_topic7
http://msdn2.microsoft.com/en-us/library/ms839674.aspx

http://msdn2.microsoft.com/fr-fr/library/ms228823(VS.80).aspx

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/windows/windowreference/windowfunctions/winmain.asp

http://msdn2.microsoft.com/en-us/library/ms838270.aspx

http://blogs.msdn.com/medmedia/archive/2006/11/13/camera-capture-performance-part1.aspx

http://msdn2.microsoft.com/en-us/library/ms940077.aspx

http://msdn2.microsoft.com/en-us/library/aa451352.aspx

-Other
---
http://www.cppfrance.com/tutoriaux/BASE-CREATION-FENETRE-API-WINDOWS_345.aspx
http://www.kickingsoftware.com/windows_mobile_game_programming/part_1.php

http://www.techheadbrothers.com/Articles.aspx/manipulation-images-vb-net-csharp-page-2

-BMP
---
http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html

http://www.wotsit.org/list.asp?search=bmp&button=GO%21

http://forum.hardware.fr/hfr/Programmation/fichier-bitmap-programmation-sujet_10902_1.htm

http://msdn2.microsoft.com/en-us/library/ms532290.aspx

http://www.koders.com/cpp/fid976B9FB8BA4BAC85DF0C64625A60D2323BD74BE8.aspx

http://www.mvps.org/user32/gditutorial.html

http://www.cppfrance.com/infomsg_EXTRACTION-CONTOUR-IMAGE-BMP_91980.aspx

http://www.codeproject.com/bitmap/gditutorial.asp?print=true

http://chgi.developpez.com/windows/image/nt/

http://msdn2.microsoft.com/en-us/library/aa923794.aspx

- Code Guru
---
www.codeguru.com

http://www.codeguru.com/cpp/g-m/bitmap/capturing/article.php/c12327/

Annexes :

Annexe : http://www.codeguru.com/

Traduction personnelle en français du code DirectShow sur Windows Mobile de CodeGuru :

http://www.codeguru.com/cpp/g-m/bitmap/capturing/article.php/c12327/

Utilisation d'une caméra avec Windows Mobile 5

Dans la partie qui suit, nous allons voir une première idée d'implémentation avec les filtres de DirectShow. L'idée est de comprendre comment avoir accès à la caméra, sur l'OS Windows Mobile 5.0, avec DirectShow

Un certain nombre d'API pour Windows Mobile 5.0 (par exemple, SHCameraCapture) rendent la tâche triviale pour un développeur d'application sur appareil mobile de pouvoir accéder à la caméra. Toutefois cette simplicité a un prix, notamment dans la flexibilité et le contrôle de l'information provenant de la caméra. La plupart du temps, utiliser directement l'API suffit à fournir une solution, mais parfois le besoin d'avoir plus de contrôle et de flexibilité dans le code et dans l'accès à la caméra et son contenu se fait ressentir. C'est dans cette optique que le DirectShow framework de Microsoft intervient.
Dans la suite, nous allons tout d'abord comprendre comment utiliser DirectShow pour accéder à la caméra, et ensuite démontrer comment construire un « Graphe de filtres » manuellement et comment manipuler les évènements de ce graphe dans le récepteur de message de l'application (application message handler).
La figure suivante dépeint les composants dans le graphe des filtres qu'il faut utiliser dans le but de réaliser une capture vidéo.

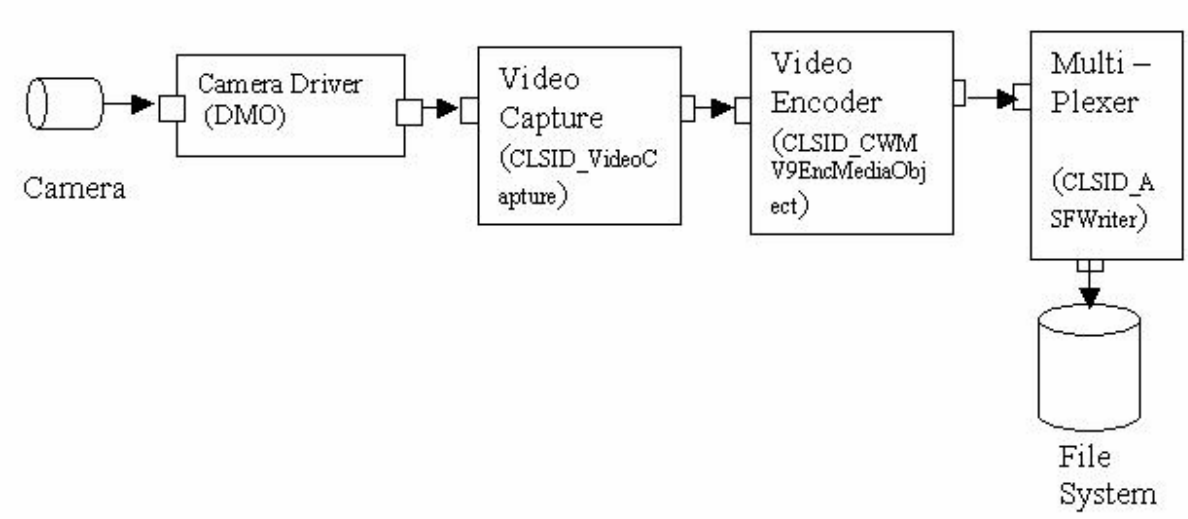


Figure : Graphe de filtres pour une capture vidéo

Enregistrement d'un flux vidéo dans un fichier .asf

La caméra est le composant hardware. Afin qu'une application puisse interagir avec la caméra, elle nécessite au préalable de parler à son driver (pilote de la caméra). Ensuite, le filtre de capture vidéo doit permettre à une application de capturer la vidéo. Après capture, il va s'agir d'encoder les données en utilisant WMV9EncMediaObject, qui est un Media Object de DirecX (DMO).
Il est possible d'utiliser un DMO à l'aide d'un DMO Wrapper Filter (filtre d'encapsulation de DMO).

Ensuite les données vidéo encodées doivent être multiplexée. Pour cela, il est possible d'utiliser un filtre d'écriture Windows Media ASF afin d'accomplir cette tâche. Ce filtre ASF multiplexe les données vidéo et l'écrit dans un fichier .asf.
Avec tout cela, le « Graphe de filtres » (filter Graph) est réalisé. Il ne s'agit maintenant uniquement de le faire tourner, à savoir l'implémenter avec du code.

Configuration de l'environnement de développement

Il faut au préalable configurer l'environnement de développement. Pour ce faire, les librairies suivantes doivent être inclues dans le "linker setting" d'un projet Smart Device de Visual Studio 2005 :

- dmoguids.lib
- strmiids.lib
- strmbase.lib
- uuid.lib

Il faut en outre aussi inclure les fichiers d'en-tête suivant (header files) dans le projet:
- atlbase.h
- dmodshow.h
- dmoreg.h
- wmcodecids.h

Note : Par souci de clarté, la gestion des erreurs n'a pas été pris en compte dans ce petit exemple. Mais une application réelle tel le "Face Tracker" va demander de gérer les exceptions et erreurs inhérentes.

Construction du Graphe

Un "graphe de filtres" qui traite la capture audio et vidéo est connu sous le nom de "Capture graph". DirectShow fournit un objet de type `CaptureGraphBuilder` qui présente une interface apellée `ICaptureGraphBuilder2`; il fournit en outre un certain nombre de méthodes qui aide à la création et au contrôle d'un graphe de capture.

En premier lieu, il va s'agir de créer les instances d'un `IGraphBuilder` et d'un `ICaptureGraphBuilder2` en utilisant la fonction COM `CoCreateInstance`:

```
HRESULT hResult = S_OK;

IGraphBuilder *pFilterGraph;
ICaptureGraphBuilder2 *pCaptureGraphBuilder;

hResult=CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC,
                         IID_IGraphBuilder,(void**)&pFilterGraph);

hResult=CoCreateInstance(CLSID_CaptureGraphBuilder, NULL,
                         CLSCTX_INPROC, IID_ICaptureGraphBuilder2,
                         (void**)& pCaptureGraphBuilder);
```

`CoCreateInstance` prend 5 paramètres :

1) Le premier représente l'ID de la classe
2) Le second décide si l'objet créé fait partie d'une agrégation
3) Le troisième spécifie le contexte dans lequel le nouvel objet créé va être présent
4) Le quatrième est une référence à l'identifiant de l'interface que l'on va utiliser pour communiquer
5) Le dernier paramètre est l'adresse de la variable qui reçoit le pointeur vers l'interface demandé

Une fois que l'on a créer les instances de `IGraphBuilder` et de `ICaptureGraphBuilder2`, nous devons appeler la méthode `SetFilterGraph` de l'interface `ICaptureGraphBuilder2`.

```
hResult = m_pCaptureGraphBuilder->SetFiltergraph( pFilterGraph );
```

Cette méthode `SetFilterGraph` prend un pointeur vers l'interface `IGraphBuilder`. Cela spécifie quel graphe le constructeur du graphe de capture va utiliser. Si nous n'appelons pas la méthode `SetFilterGraph`, le constructeur du Capture graph va automatiquement en créer un quand il aura besoin.

A présent, nous sommes prêt à créer une instance du filtre de capture vidéo.
Le code suivant initialise un filtre de capture "Video" à l'aide du pointeur qui est retourné par `CoCreateInstance`.

```
IBaseFilter *pVideoCaptureFilter

hResult=CoCreateInstance(CLSID_VideoCapture, NULL, CLSCTX_INPROC,
                         IID_IBaseFilter, (void**)&pVideoCaptureFilter);
```

Il faut avoir également un pointeur vers `IPersistPropertyBag` depuis le filtre de capture vidéo. Ce pointeur est utilisé pour permettre au filtre de capture d'accéder au dispositif de capture (en d'autre termes, le dispositif, ou device, est la caméra) :

```
IPersistPropertyBag *pPropertyBag;

hResult=pVideoCaptureFilter->QueryInterface( &pPropertyBag );
```

Il s'agit maintenant d'avoir une emprise sur la caméra que l'on va utiliser pour capturer la vidéo. Il va donc falloir énumérer les camera devices disponibles en utilisant les fonctions `FindFirstDevice` et `FindNextDevice`. Il se peut que plusieurs caméras soient présentes sur le device en question. (HTC Universal est un exemple). Afin de garder le code simple pour cet exemple, nous allons utiliser la fonction `FindFirstDevice` afin d'avoir la première caméra disponible sur le device:

```
DEVMGR_DEVICE_INFORMATION devInfo;
CComVariant  CamName;
CPropertyBag PropBag;

GUID guidCamera = { 0xCB998A05, 0x122C, 0x4166, 0x84, 0x6A, 0x93,
                    0x3E, 0x4D, 0x7E, 0x3C, 0x86 };

devInfo.dwSize = sizeof(devInfo);

FindFirstDevice( DeviceSearchByGuid, &guidCamera, & devInfo);

CamName=devInfo.szLegacyName

PropBag.Write( _T("VCapName"), &CamName );

pPropertyBag->Load( &PropBag, NULL );

hResult =pFilterGraph->AddFilter( pVideoCaptureFilter,
                                  _T("Video Capture Filter") );
pPropertyBag.Release();
```

Il est à noter le premier paramètre de la fonction `FindFirstDevice` : `DeviceSearchByGuid`. Il spécifie le type de recherche. Il y a d'autres options de recherches tels `DeviceSearchByLegacyName`, `DeviceSearchByDeviceName`, etc.

Mais `DeviceSearchByGuid` est la méthode plus facile pour trouver un dispositif de capture (capture device).
L'information concernant le device est retournée dans la structure `DEVMGR_DEVICE_INFORMATION`. Il faut enregistre la valeur de `szLegacyName` dans une variable `CComVariant`, et pour cela il faut disposer d'un objet qui implémente l'interface `IPropertyBag`.

Dans l'exemple du code on peut voir une classe de base `CPropertyBag` qui implémente `IPropertyBag`. Cet objet est nécessaire pour passer le nom du capture device au filtre. Le string `VCapName` identifie la propriété du filtre pour le nom du vidéo capture device. Une fois que nous avons accès au capture device, il est possible d'ajouter le filtre de capture "Vidéo" au graphe des filtres. Pour cela, il faut utiliser la méthode `AddFilter` du graph manager. Cette méthode prend 2 paramètres : le premier est un pointeur vers le filtre qui doit être ajouter, et le deuxième est le nom du filtre. Le deuxième paramètre peut prendre la valeur NULL; dans ce cas, le graph manager de filtre génère un nom unique pour le filtre. Si nous avons fourni un nom qui provoque un conflit avec d'autres filtre, le manager va modifier le nom pour le rendre unique.

Il faut instancier l'encodeur WMV9 :

```
IBaseFilter *pVideoEncoder;
IDMOWrapperFilter *pWrapperFilter;

hResult=CoCreateInstance(CLSID_DMOWrapperFilter, NULL,CLSCTX_INPROC,
                         IID_IBaseFilter, (void**)&pVideoEncoder);

hResult =pVideoEncoder->QueryInterface( &pWrapperFilter );

hResult =pWrapperFilter->Init( CLSID_CWMV9EncMediaObject,
                               DMOCATEGORY_VIDEO_ENCODER );

hResult=pFilterGraph->AddFilter( pVideoEncoder, L"WMV9DMO Encoder");
```

Etant donné le fait que l'encodeur WMV9 est un DMO, il n'est pas possible de l'ajouter/l'utiliser à la manière d'autres filtres. Mais DirectShow fournit un filtre d'encapsulation (wrapper filter) qui permet d'utiliser une DMO comme tout autre filtre. Il faut pour cela créer une instance du filtre DMO encapsulant et ensuite initialiser l'encodeur WMV9 DMO avec ce dernier.
Après l'initialisation de la DMO, il faut le rajouter dans le graphe des filtres de la manière suivante :

```
IBaseFilter *pASFMultiplexer;
IFileSinkFilter *pFileSinkFilter;

hResult = pCaptureGraphBuilder->SetOutputFileName(
   &MEDIASUBTYPE_Asf, T("\\test.asf"), &pASFMultiplexer,
   &pFileSinkFilter );
```

Nous avons ajouter la source et le filtre de transformation dans le filtre de graphe, la dernière chose à rajouter et par conséquence à mettre en place est une sortie (ce qui peut être rendu possible à l'aide d'un "sink filter"). Pour ce faire, nous employons la méthode `SetOutputFileName` de `lCaptureGraphBuilder2`.
Le premier paramètre est un sous-type de média; le deuxième paramètre est le nom du fichier dans lequel nous souhaitons sauver la vidéo; le troisième paramètre est l'adresse du pointeur qui reçoit l'interface multiplexée; et enfin le dernier paramètre reçoit l'interface de l'écriture de fichier.

Avec tout cela, le graphe de filtres est prêt. La dernière chose qu'il s'agit de réaliser est de connecter le filtre source, l'encodeur et le multiplexeur.
Cela est rendu possible en utilisant la méthode `RenderStream` du constructeur de graphe de la manière suivante:

```
hResult = pCaptureGraphBuilder->RenderStream( &PIN_CATEGORY_CAPTURE,
                                              &MEDIATYPE_Video,
                                              m_pVideoCaptureFilter,
                                              pVideoEncoder,
                                              pASFMultiplexer );
```

Le premier paramètre représente la catégorie de la pin (qui peut être `NULL` si nous souhaitons qu'elle corresponde à toutes les catégories)

Le deuxième paramètre spécifie le type de média. Les troisièmes, quatrièmes et cinquièmes paramètres spécifient respectivement le filtre de démarrage, le filtre intermédiaire et le filtre de sortie finale (sink filter). La méthode connecte le filtre source au filtre de transformation et ensuite le filtre de transformation au filtre de sortie (sink filter).

Maintenant le graphe est réalisé, nous pouvons commencer à capturer la vidéo.

Contrôle du graphe

Avant de pouvoir capturer la vidéo, nous avons besoin de deux choses encore :
Les pointeurs `IMediaEventEx` et `IMediaControl`.

`IMediaEventEx` dérive de `IMediaEvent`, et supporte la notification des évènements en provenance du graphe des filtres et des filtres individuels vers l'application. `IMediaEventEx` fournit une méthode vers la fenêtre d'enregistrement qui reçoit un message lorsqu'un évènement survient.

`IMediaControl` représente une interface exposée par le graphe de filtres et qui autorise une application à contrôler le streaming média à travers ce graphe. L'application peut donc l'utiliser pour « démarrer », « arrêter », « mettre en pause » le graphe courant.

L'exemple de code suivant montre comment questionner le graphe de filtre pour son interface `IMediaEventEx`. Une fois reçu le pointeur sur l'interface `IMediaEventEx`, il appelle ensuite sa méthode `SetNotifyWindow`, en lui passant un « handle » (emprise) vers la fenêtre qui elle-même possède une emprise (« handle ») sur le message. Le deuxième paramètre est le message qui sera passé comme une notification au Windows message handler. Le troisième paramètre est l'instance data (qui peut avoir la valeur 0 aussi) :

```
IMediaEventEx *pMediaEvent;
IMediaControl *pMediaControl;

#define WM_GRAPHNOTIFY WM_APP+1

hResult =pFilterGraph->QueryInterface( IID_IMediaEventEx, (void**)
                                        &pMediaEvent );

hResult =pMediaEvent->SetNotifyWindow((OAHWND)hWnd,
                                      WM_GRAPHNOTIFY,0);

hResult=pFilterGraph->QueryInterface(&pMediaControl);

hResult =pMediaControl->Run();
```

Lorsqu'un évènement survient, DirectShow va envoyer un `WM_GRAPHNOTIFY` à la fenêtre spécifiée.

Remarque : `WM_GRAPHNOTIFY` est utilisé à titre d'exemple. Nous aurions pu tout aussi bien avoir un autre message application.

L'étape suivante est d'avoir un pointeur vers l'interface `IMediaControl`. Cette interface est importante car elle permet de contrôler le graphe.
En effet, il suffit d'appeler sa méthode `Run` pour mettre le graphe tout entier dans un état « running ». Le code suivant montre comment démarrer et stopper la capture en utilisant la méthode `ControleStream` de `CaptureGraphBuilder` :

```
LONGLONG dwStart  = 0, dwEnd      = 0;
WORD wStartCookie = 1, wEndCookie = 2;

dwEnd=MAXLONGLONG

//start capturing
hResult=pCaptureGraphBuilder->ControlStream(
   &PIN_CATEGORY_CAPTURE, &MEDIATYPE_Video,
   pVideoCaptureFilter, &dwStart, &dwEnd,
   wStartCookie, wEndCookie );

//Stop capturing
dwStart=0;
hResult=pFilterGraph->QueryInterface(&pMediaSeeking );

hResult=pMediaSeeking->GetCurrentPosition( &dwEnd );
```

```
hResult= pCaptureGraphBuilder->ControlStream(
   &PIN_CATEGORY_CAPTURE, &MEDIATYPE_Video, pVideoCaptureFilter,
   &dwStart, &dwEnd, wStartCookie, wEndCookie );
```

Ce code utilise le critère de recherche fourni dans l'appel de méthode permettant de localiser une pin de sortie sur le filtre de capture. `ControlStream` autorise une application à contrôler des streams (flux) sans que l'application ait besoin d'énumérer des filtres et des pins dans le graphe.

`Start` et `End` spécifient les temps de démarrage et d'arrêt, équivalent au start et stop habituels (Il est à remarquer que `MAX_LONGLONG` est la valeur la plus grande pour le temps de référence). Au démarrage, le `End` est mis à la valeur `MAXLONGLONG`. Lorsque nous souhaitons stopper, il faut d'abord récupérer la position courante du stream en utilisant la méthode `GetCurrentPosition` de l'interface `IMediaSeeking`. Il va ensuite falloir apeller la méthode `ControlStream` avec `Start` mis à 0 et `End` mis à la valeur de la position actuelle.

Nous avons maintenant le graphe prêt et de plus il est lancé. Il est maintenant possible de l'utiliser pour capturer et enregistrer dans un fichier .asf.

Prise en main des évènements du graphe (Graph Events)

Etant donné qu'une application va contrôler le graphe, il nous faut écrire le code qui faciliter cette prise en main.
Nous avons jusque là déjà enregistrer la fenêtre et le message dans le graphe de filtres. La dernière étape qu'il nous reste à réaliser concerne la prise en main est de passer le message au window message handler.

Ceci se fait de la manière suivante:

```
BOOL CALLBACK VidCapDlgProc(HWND hDlg,UINT Msg,WPARAM wParam,
                            LPARAM lParam)
{
   ... ... ... ...
   case WM_GRAPHNOTIFY:
      {
         ProcessGraphMessage();
      }
   ... ... ... ...
}

ProcessGraphMessage()
{
   HRESULT hResult=S_OK;
   long leventCode, param1, param2;
```

```
   while(hResult=pEvent->GetEvent(&leventCode, &param1, &param2, 0),

   SUCCEEDED(hResult))
   {

      hResult = pEvent->FreeEventParams(leventCode, param1, param2);
      if (EC_STREAM_CONTROL_STOPPED == leventCode)
      {
         pMediaControl->Stop();
         break;
      }
      else if(EC_CAP_FILE_COMPLETED== leventCode)
      {
         //Handle the file capture completed event
      }
      else if(EC_CAP_FILE_WRITE_ERROR== leventCode)
      {
         //Handle the file write error event
      }
   }

}
```

Nous passons le message `WM_GRAPHNOTIFY` au windows handler. DirectShow envoie ce message à l'application lorsque n'importe quel évènement surgit. L'application appelle une méthode définie pour l'utilisateur pour traiter les évènements. La méthode `GetEvent` de l'interface `IMediaEvent` recherche le code de l'évènement et deux paramètres d'évènement de la queue.

Etant donné que le message loop et le event notification (notification d'évènements) sont asynchrones, il est très probable que la queue contienne plus qu'un évènement. Par conséquent, le code de `GetEvent` est appelé dans la boucle (loop) jusqu'à ce qu'il renvoie un code erroné. De plus chaque fois que nous appelons `GetEvent`, il est important d'appeler `FreeEvent` afin de libérer la ressource associée au paramètre de l'évènement.

En bon programmeur, il ne faut de plus pas oublier de libérer les ressources à la fin. Ceci se fait en appelant la méthode `Release` sur chaque objet qui a été créé, de la manière suivante :

```
PVideoCaptureFilter->Release ();
pVideoEncoder->Release ();
pMediaEvent ->Release();
pMediaSeeking ->Release();
pASFMultiplexer->Release();
pFileSinkFilter->Release();
pWrapperFilter ->Release();
pFilterGraph->Release();
pCaptureGraphBuilder->Release();
```

A travers cette explication de capture et enregistrement d'une vidéo dans un fichier, l'objectif principal a été de comprendre comment manuellement créé, démarré, et contrôler un graphe de filtres. Selon Microsoft, DirectShow Framework permet d'avoir un bon contrôle dans une application où il s'agit de faire une capture depuis la caméra.

Annexe : Parties de code commentés de DirectShow

```
/*****************************************************************************************************/
/*** Function setResolutionCamera(HANDLE hMyCamera, DWORD dwID )                                   ***/
/*****************************************************************************************************/
/*** This function is not used but implemented                                                     ***/
/*** (it could help and set the best possible camera resolution on a device )                      ***/
/***                                                                                              ***/
/*** 2 parameter  : 1) the handler on the camera)                                                  ***/
/***                2) the desired resolution                                                     ***/
/*** return value : HRESULT (if problems encountered while of setting the resolution)            ***/
/*****************************************************************************************************/
HRESULT faceTrackerImp::setResolutionCamera(HANDLE hMyCamera, DWORD dwID )
{
    HRESULT hResult = S_OK ;
    //Instance of the structure CameraFT
    CameraFT *pCameraFT = (CameraFT *)hMyCamera ; //type casting the handler

    //IAMStreamConfig interface helps us to set the resolution of the camera
    CComPtr<IAMStreamConfig> pIAMStreamConfig ;

    hResult = pCameraFT->pCaptureGraphBuilder->FindInterface( &PIN_CATEGORY_STILL,
                                                              &MEDIATYPE_Video,
                                                              pCameraFT->pVideoCaptureFilter,
                                                              IID_IAMStreamConfig,
                                                              (void **) &pIAMStreamConfig ) ;
    // variable declarations

    //number of VIDEO_STREAM_CONFIG_CAPS and/or AUDIO_STREAM_CONFIG_CAPS structures supported
    int count   = 0 ;
    //size of the configuration structure, AUDIO_STREAM_CONFIG_CAPS or VIDEO_STREAM_CONFIG_CAPS
    int size    = 0 ;

    //retrieving the number of stream capabilities structures for the compressor
    hResult = pIAMStreamConfig->GetNumberOfCapabilities(& count,& size) ;
    if(SUCCEEDED(hResult) && size == sizeof(VIDEO_STREAM_CONFIG_CAPS) && (dwID < (DWORD)count))
    {
        //VIDEO_STREAM_CONFIG_CAPS is a structure with information on possible
        //connections & enables to chose frame rate, etc...
        VIDEO_STREAM_CONFIG_CAPS vscc ;

        //AM_MEDIA_TYPE gives information about a media sample -> it's format
        AM_MEDIA_TYPE *pMediaTypeconfig ;

        //getting audio, video, or other capabilities of a stream depending on which type of structure
        //is pointed. Here pointed structure is our  VIDEO_STREAM_CONFIG_CAPS of course... ;)
        hResult = pIAMStreamConfig->GetStreamCaps(dwID, &pMediaTypeconfig, (BYTE *)&vscc) ;
        if(SUCCEEDED(hResult))
        {
            //setting the audio or video stream's format
            pIAMStreamConfig->SetFormat(pMediaTypeconfig) ;

            //deletes allocated memory for the AM_MEDIA_TYPE structure
            DeleteMediaType(pMediaTypeconfig) ;
        }
        else
        {
            hResult = E_FAIL ;
        }

    }

    return hResult ;
}
```

```
/*****************************************************************************************************/
/*** Function launchCapture(HANDLE handleCamera)                                                   ***/
/*****************************************************************************************************/
/*** This function launches the preview by setting the MediaControl with "RUN" mode                ***/
/*** (!  be aware that the connections should be put up first!                                      ***/
/***     with the cameraConnectAll(...) function )                                                 ***/
/***                                                                                               ***/
/*** 1 parameter  : 1) the handler on the camera)                                                  ***/
/*** return value : HRESULT (specifies if launching the capture went fine or not)                  ***/
/*****************************************************************************************************/
HRESULT faceTrackerImp::launchCapture(HANDLE handleCamera)
{
    HRESULT hResult ;

    CameraFT *pCameraFT = (CameraFT *) handleCamera ;

    //verify if the graph is running (if graph is not running then launch it !)
    if(pCameraFT->bGraphRunning == 0)
    {
        //Running the graph
        hResult = pCameraFT->pMediaControl->Run() ;


        hResult = pCameraFT->pVideoCaptureFilter.QueryInterface( &pCameraFT->pVideoControl ) ;
        if(FAILED(hResult))
        {
            MessageBox (NULL, _T("ERROR:pVideoControl queryInterface!"),
                        _T("ERROR"), MB_OK);
        }
        hResult = pCameraFT->pCaptureGraphBuilder->FindPin( pCameraFT->pVideoCaptureFilter ,
                                                            PINDIR_OUTPUT ,
                                                            &PIN_CATEGORY_STILL,
                                                            &MEDIATYPE_Video,
                                                            FALSE,
                                                            0,
                                                            &pCameraFT->pStillPin);
        if(FAILED(hResult))
        {
            MessageBox (NULL, _T("ERROR:PROBLEM Finding Pin!"),
                        _T("ERROR"), MB_OK);
        }


        pCameraFT->bGraphRunning = 1 ; //specify that the graph is running

        pCameraFT->bGraphRunning = 1 ; //specify that the graph is running


    }

    return hResult ;
}
```

```
/*****************************************************************************************************/
/*** Function stopAndQuit(HANDLE handleCamera)                                                     ***/
/*****************************************************************************************************/
/*** This function stops and quit the whole preview application by "killing" the filter graph     ***/
/***                                                                                               ***/
/*** 1 parameter  : 1) the handler on the camera)                                                  ***/
/*** return value : HRESULT (specifies if stopping&quitting the capture went fine or not)         ***/
/*****************************************************************************************************/
HRESULT faceTrackerImp::stopAndQuit(HANDLE handleCamera)
{
    HRESULT hResult = S_OK ;

    CameraFT *pCameraFT = (CameraFT *) handleCamera ;

    if(gpVideoWindow)
    {
        gpVideoWindow->put_Visible(OAFALSE) ;

        //important! stop the owner of pVideoWindow, or the video renderer
        //still believes in having a valid parent window
        gpVideoWindow->put_Owner(NULL) ;
        gpVideoWindow->Release() ;

    }

    pCameraFT->pMediaControl->Stop() ;
    pCameraFT->pMediaControl->Release() ;
    pCameraFT->pVideoCaptureFilter.Release() ;
    pCameraFT->pCaptureGraphBuilder.Release() ;
    pCameraFT->pFilterGraphBuilder.Release() ;
    CoUninitialize() ;

    delete pCameraFT ; //free memory

    return hResult ;

}
```

```
/*****************************************************************************************************/
/*** Function ForegroundPreview(BOOL Focus, HANDLE handleCamera)                                   ***/
/*****************************************************************************************************/
/*** This function puts the video window in foreground                                             ***/
/***                                                                                               ***/
/*** 2 parameters : 1) the boolean for the focus                                                   ***/
/***                2) the handler on the camera                                                   ***/
/*** return value : -                                                                              ***/
/*****************************************************************************************************/
VOID faceTrackerImp::ForegroundPreview(BOOL Focus, HANDLE handleCamera)
{
    CameraFT *pCameraFT = (CameraFT *) handleCamera ;
    if (Focus)
    {
        gpVideoWindow->SetWindowForeground(OATRUE);
        gpVideoWindow->put_Visible(OATRUE) ;
    }
    else
    {
        gpVideoWindow->SetWindowForeground(OAFALSE);
    }
}
```

```
/*****************************************************************************************************/
/*** Function takePhoto(HANDLE handleCamera)                                                       ***/
/*****************************************************************************************************/
/*** This function takes a photo by "stealing" a frame from the video stream                       ***/
/***                                                                                              ***/
/*** 1 parameter  : 1) the handler on the camera                                                   ***/
/*** return value : HRESULT (specifies if taking a photo went fine or not)                         ***/
/*****************************************************************************************************/
HRESULT faceTrackerImp::takePhoto(HANDLE handleCamera)
{

    HRESULT hResult = S_OK ;
    CameraFT *pCameraFT = (CameraFT *) handleCamera ;

    //save in file&trigger to take the photo
    //instead of NULL a media type could be specified
    hResult = pCameraFT->pFileSink->SetFileName( L"\\FTtest.bmp",NULL ) ;

    hResult = pCameraFT->pVideoControl->SetMode(pCameraFT->pStillPin,VideoControlFlag_Trigger) ;


    if(FAILED(hResult))
    {
        MessageBox (NULL, _T("ERROR:taking photo& triggering queryInterface problem!"),
                    _T("ERROR"), MB_OK);
    }


    return hResult ;

}
```

Les variables globales suivantes ont été déclarées :

```
/*** Including header files from functions to be exported to managed code ***********************/

#include "FaceTrackerImp.h"

/*** global variable Declaration ************************************************************/
HWND gTheHwnd = NULL;
HANDLE gTheHandle = NULL;

int geventcode;
BOOL running;
BOOL newevent;

faceTrackerImp *myFaceTrackerImp = new faceTrackerImp() ;
```

Variables globales permettant d'implémenter l'astuce

Regardons de plus près le code de la fonction previewDisplay(HWND pbhandle)

```
/*****************************************************************************************************/
/*** Function previewDisplay(HWND pbhandle)                                                        ***/
/*****************************************************************************************************/
/*** This function is a DLL exported function                                                      ***/
/*** it contains a while loop in order to be called forever for the Preview Window                 ***/
/***                                                                                              ***/
/*** 1 parameter  : 1) the handler on the pictureBox(in the C# GUI)                                ***/
/*** return value : -                                                                              ***/
/*****************************************************************************************************/
extern "C" __declspec(dllexport)void __stdcall previewDisplay(HWND pbhandle)
{
    running = true ;
    HRESULT hResult;

    DWORD theDword = 0 ;
    //connecting camera, filters & all DirectShow's stuff, THEN launch the capture
    hResult = myFaceTrackerImp->cameraConnectAll(gTheHwnd, &gTheHandle, theDword,pbhandle) ;
    hResult = myFaceTrackerImp->launchCapture(gTheHandle) ;

    //forever loop for the preview
    while(running)
    {
        if(newevent)
        {
            switch(geventcode)
            {
            case 1:
                running = false;
            case 2:

                myFaceTrackerImp->takePhoto(gTheHandle) ;
                //myFaceTrackerImp->ForegroundPreview(TRUE, gTheHandle) ;
            case 3 :
                myFaceTrackerImp->updatePreview(pbhandle);


            }

            newevent = false;
        }
        Sleep(1000);
    }

    myFaceTrackerImp->stopAndQuit(gTheHandle) ;

}
```

Fonctions internes à DirectShow

Un nouvel évènement est arrivé

On switch la fonction interne à appeler en fonction de « geventcode » qui change en fonction de la fonction demandé par le GUI

Voici la fonction interne à ce fichier qui implémente notre gestionnaire d'évènement :

Ainsi les 3 autres fonctions exposées n'auront qu'à appeler cette fonction interne en spécifiant un eventcode différent comme paramètre qui sera ensuite copié dans la variable globale geventcode.

```
/*****************************************************************************************************/
/*** Function fnewevent(int eventcode)                                                             ***/
/*****************************************************************************************************/
/*** This function is a sort of "cheat" but "clean" mode to access into the DLL                    ***/
/*** and trigger to the functions we want to execute                                               ***/
/***                                                                                              ***/
/*** 1 parameter  : 1) an integer   specifing the DLL inside functions to execute                  ***/
/*** return value : -                                                                              ***/
/*****************************************************************************************************/
void  fnewevent(int eventcode)
{
    //we trigger the global event
    newevent = true;
    geventcode = eventcode;

}
```

La fonction exposée au GUI updatePreviewNow() permet de créer une nouvelle videoWindow. Et nous voyons ici qu'elle fait appel à la fonction interne fnewevent(int eventcode) en passant un eventCode lui permettant d'appeler les fonctions internes lui concernant dans la boucle infinie principale.

```
/*****************************************************************************************************/
/*** Function updatePreviewNow()                                                                   ***/
/*****************************************************************************************************/
/*** This function is a DLL exported function                                                      ***/
/*** it contains a call to the new event function to lauch a new preview                           ***/
/*** (used by GUI for the small preview window)                                                    ***/
/***                                                                                              ***/
/*** 0 parameter  : -                                                                              ***/
/*** return value : -                                                                              ***/
/*****************************************************************************************************/
extern "C" __declspec(dllexport)void __stdcall updatePreviewNow()
{
    fnewevent(3);
}
```

La fonction exposée au GUI stop() permet de stopper la capture vidéo. Et nous voyons ici qu'elle fait appel à la fonction interne fnewevent(int eventcode) en passant un eventCode lui permettant d'appeler les fonctions internes lui concernant dans la boucle infinie principale.

```
/*****************************************************************************************************/
/*** Function stop()                                                                               ***/
/*****************************************************************************************************/
/*** This function is a DLL exported function                                                      ***/
/*** it contains a call to the new event function to stop the preview and quit it's loop           ***/
/*** (used by GUI to kill all DirectShow stuff if needed)                                          ***/
/***                                                                                              ***/
/*** 0 parameter  : -                                                                              ***/
/*** return value : -                                                                              ***/
/*****************************************************************************************************/
extern "C" __declspec(dllexport)void __stdcall stop()
{

    fnewevent(1);

}
```

La fonction exposée au GUI takePicture() permet de prendre une image du flux vidéo. Et nous voyons ici qu'elle fait appel à la fonction interne fnewevent(int eventcode) en passant un eventCode lui permettant d'appeler les fonctions internes lui concernant dans la boucle infinie principale.

```
/*****************************************************************************************************/
/*** Function takePicture()                                                                        ***/
/*****************************************************************************************************/
/*** This function is a DLL exported function                                                      ***/
/*** it contains a call to the new event function to take a picture                                ***/
/*** (used by GUI to "steal" from the video stream)                                                ***/
/***                                                                                              ***/
/*** 0 parameter  : -                                                                              ***/
/*** return value : -                                                                              ***/
/*****************************************************************************************************/

extern "C" __declspec(dllexport)void __stdcall takePicture()
{

    fnewevent(2);
    while(newevent)
    {
        Sleep(50); //sleep added in case pictures are taken too much often...=>problem!
    }


}
```

Annexe : Parties de code expliqués sur l'API FaceFindingAPI_0.2.2

```
/*****************************************************************************************************/
/*** Function ppm_loadbmp ( char *filename,                                                        ***/
/***                        int *image_width,                                                      ***/
/***                        int *image_height,                                                     ***/
/***                        unsigned char **image)                                                 ***/
/*****************************************************************************************************/
/*** This function is an internal function used to load a BMP file (part of the code is            ***/
/*** inspired by the examples given with IDIAP's library)                                          ***/
/*** This function is used by the exported function track()                                        ***/
/***                                                                                              ***/
/***                                                                                              ***/
/*** 4 parameters : 1) the path to the .bmp file to track a face(string value)                     ***/
/***                2) a pointer to an int in order to set the image width                        ***/
/***                3) a pointer to an int in order to set the image height                       ***/
/***                4) a pointer to an int in order to allocate the needed memory for the image***/
/*** return value : ff_image_type( the newly loaded image)                                         ***/
/*****************************************************************************************************/
ff_image_type ppm_loadbmp (char *filename,
                           int *image_width,
                           int *image_height,
                           unsigned char **image)
{
    //local variable declarations
    unsigned char *imageline = NULL;
    char tmp_line [255];
    FILE *image_fd;
    int nb_bytes;
    char *fname = filename;

    //FaceFindingAPI_0.2.2 variable declaration
    ff_image_type image_type;

    //Bitmap sections
    BITMAPFILEHEADER bmfh;
    BITMAPINFOHEADER bmih;

    //Opening image file
    image_fd = fopen (fname, "rb");
```

Ouverture du fichier à image

```
    //Image file could not be opened
    if (!image_fd)
    {
        /* problem opening the file */
        printf("filename %s",fname);
        printf("filename %s",filename);

        //  error filname : perror (filename);
        return image_type;
    }

    //Image is correctly loaded
    printf("loaded\n");

    //Reading information header & file header
    fread(&bmfh , sizeof(BITMAPFILEHEADER) , 1 , image_fd) ;
    fread(&bmih , sizeof(BITMAPINFOHEADER) , 1 , image_fd) ;

    //assuming the BMP file is 24 bits bmp file , nbyte = 4
    nb_bytes = bmih.biBitCount/8;

    image_type = FF_IMAGE_TYPE_RGB24;

    //for debugging in order to be sure we have a 24 bits bmp file
    printf("Bits per pixel = %d\n", bmih.biBitCount) ;

    //setting the reference parameters with the image width and height
    *image_width = bmih.biWidth;
    *image_height = bmih.biHeight;

    //allocate the space for the image, and load it
    *image =(unsigned char*) malloc (*image_width * *image_height * nb_bytes * sizeof (char));

    imageline  =(unsigned char*) malloc (*image_width * nb_bytes * sizeof(char));

    // -- bmp file rows are padded to multiples of 4 bytes in 24 bmp files
    int pad = (((bmih.biWidth * 3) % 4) == 0) ? 0 : 4 - ((bmih.biWidth * 3) % 4);


    //read the real image
    //The important thing here is to read the rows and invert them to respect
    //bmp file structure
    for (int t = (*image_height)-1; t >=0;--t)
    {
            fread(*image + (nb_bytes * t*(*image_width)),
                  sizeof(char),*image_width*nb_bytes,image_fd );

            fread(imageline,pad,1,image_fd);
    }

    //free memory
    free(imageline);

    //Ok! image file can be closed now
    fclose (image_fd);

    //returns the newly loaded image
    return image_type;
}
```

Nous supposons que nous avons une image RGB 24bits et nous récupérons la hauteur et largeur de l'image.

Allocation mémoire nécessaires

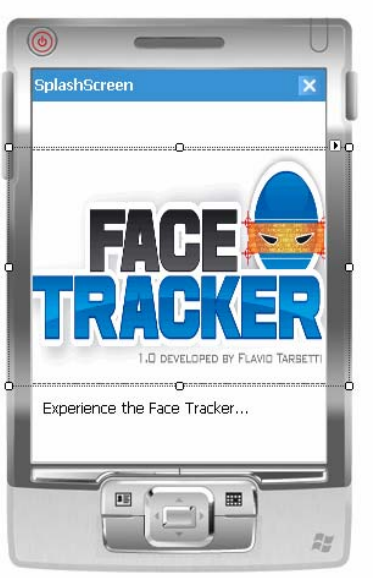
Lecture de l'image

Annexe : Parties de codes expliqués sur le projet GUI

Windows Form SplashScreenImage.cs

Afin de rendre l'application plus attrayante, un logo a été créé et implémenter comme « SplashScreen » (ou image de bienvenue) au démarrage du système.

Ce windows form `SplashScreenImage.cs` comprend un designer:



Ce logo va constituer le logo de l'application « Face Tracker ».

Le code de cette application est assez simple :

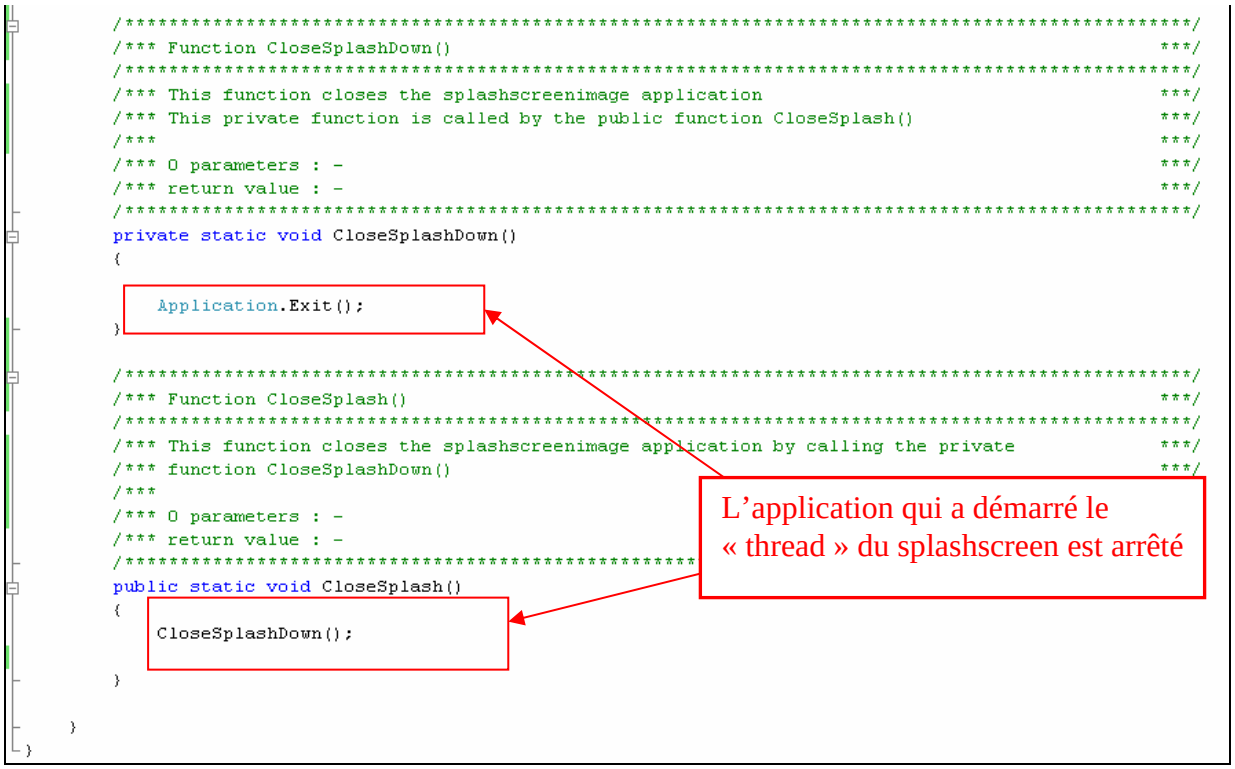
Nous allons uniquement voir les parties importantes du code. (le code étant assez bien documenter pour plus de précisions)

```
/*****************************************************************************************************/
/*!
 *  \file
 *      SplashScreenImage.cs in FT_GUI -
 *  FACETRACKER ON WINDOWS MOBILE BASED PHONE / This file is used to create all the functions
 *                                              needed to control splashscreen image at the
 *                                              application startup
 *
 *  \date       November 2007
 *  \author     Flavio Tarsetti (flavio dot tarsetti at mycable dot ch)
 *                              (tarsflav at students dot hevs dot ch )
 *                              flavio.tarsetti@mycable.ch  or tarsflav@students.hevs.ch
 */
/*****************************************************************************************************/

/*** References used  ********************************************************************************/

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;

/*** namespace specification *************************************************************************/
namespace FT_FaceTracker
{
    public partial class SplashScreenImage : Form
    {
        //private variables
        private static Thread _splashLauncher;
        private static SplashScreenImage _splashScreen;

        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;otherwise, false.</
```

Déclaration du « thread » pour le splashScreen et d'une image

```
protected override void Dispose(bool disposing)
{
if (disposing && (components2 != null))
{
    components2.Dispose();
}
base.Dispose(disposing);
}

Windows Form Designer generated code

private SplashScreenImage()
{
     InitializeComponent();
}

/*****************************************************************************************************/
/*** Function ShowSplash()                                                                         ***/
/*****************************************************************************************************/
/*** This function is used to start the thread that puts up the splashscreen image                ***/
/*** it calls the private function LaunchSplash() in order to create the image and see it          ***/
/***                                                                                               ***/
/*** 0 parameters : -                                                                              ***/
/*** return value : -                                                                              ***/
/*****************************************************************************************************/
public static void ShowSplash()
{
   //Show the form in a new thread
   _splashLauncher = new Thread(new ThreadStart(LaunchSplash));
   _splashLauncher.IsBackground = true;
   _splashLauncher.Start();

}
/*****************************************************************************
/*** Function LaunchSplash()
/*****************************************************************************
/*** This function creates an image and puts it up on screen
/***
/*** 0 parameters : -
/*** return value : -
/*****************************************************************************
private static void LaunchSplash()
{
     _splashScreen = new SplashScreenImage();

    //Create new message pump
    Application.Run(_splashScreen);
}
```

Lancement de l'application qui va démarrer le « thread » s'occupant du splashscreen



Ce bout de code est totalement indépendant du GUI principale, il n'a été ajouté là que dans un souci d'esthétisme.

Program.cs

Ce fichier comprend le point d'entrée de l'application. En effet c'est là que se trouve la fonction main().
Elle n'a pour but que de :

- lancer une fenêtre de bienvenue
- fermer la fenêtre en question
- démarrer l'application principale

```
/*****************************************************************************************************/
/*!
 *  \file
 *      Program.cs in FT_GUI -
 *  FACETRACKER ON WINDOWS MOBILE BASED PHONE / This file has the main entry point of the
 *                                              application. It launches a splashscreen image
 *                                              before launching the application.
 *
 *  \date       November 2007
 *  \author     Flavio Tarsetti (flavio dot tarsetti at mycable dot ch)
 *                              (tarsflav at students dot hevs dot ch )
 *                              flavio.tarsetti@mycable.ch  or tarsflav@students.hevs.ch
 */
/*****************************************************************************************************/

/*** References used  ********************************************************************************/
using System;
using System.Collections.Generic;
using System.Windows.Forms;

/*** namespace specification *************************************************************************/
namespace FT_FaceTracker
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [MTAThread]
        static void Main()
        {
            //Spash Screen Image is started before the main application
            SplashScreenImage.ShowSplash();
            System.Threading.Thread.Sleep(5000);

            //Close the splash
            SplashScreenImage.CloseSplash();

            //launching the "Face Tracker" application
            Application.Run(new FT_pb());
        }

    }
}
```

- Ouverture fenêtre de bienvenue
- Fermeture fenêtre de bienvenue
- démarrage application « Face Tracker »

FT.cs

Ce Windows Form FT.cs comprend un Designer :

Nous voyons ici le GUI principale avec lequel l'utilisateur va interagir :

Annexe :
Portage de la librairie FaceFinding_API_0.0.2

Le grand défit a été de porter cette librairie sur un système *MS Windows Mobile*.

Il faut savoir que tous les fichiers de cette librairie ont été intégrés dans le portage du Face Tracker sur le système Windows Mobile.

Cependant, un certain nombre de changements ont dû être effectué dans le code de la librairie « FaceFindingAPI_0.2.2 ».

Ce chapitre regroupe toutes les modifications apportées au code fournis par l'IDIAP en montrant à chaque fois le code original et les modifications apportées.

Modification des fichiers d'en-tête (Header Files)

Fichier `ff_api.h` :

Contrairement à l'ancienne librairie, cette nouvelle librairie n'exposait pas les fonctions spécifiant les coordonnées des yeux, il a donc fallut les exposer :

La position des yeux sur une image est spécifiée suivant les coordonnées (x,y) en pixels.

Le code suivant a été rajouté :

Modifications apportées :
Lignes 118 à 123 :

```
//eye information
int xeyel;
int yeyel;
int xeyer;
int yeyer;
```

Modification des fichiers sources (Source Files) :

Fichier `ff_api.cpp` :

Contrairement à l'ancienne librairie, cette nouvelle librairie n'exposait pas les fonctions spécifiant les coordonnées des yeux, il a donc fallut les exposer :

La position des yeux sur une image est spécifiée suivant les coordonnées (x,y) en pixels.

Le code suivant a été rajouté :

Modifications apportées :
Ligne 570 à 575 :

```
faceinfo->xeyel= finder->tracker->faces[number]->eyeL_x ;
faceinfo->yeyel= finder->tracker->faces[number]->eyeL_y  ;
faceinfo->xeyer = finder->tracker->faces[number]->eyeR_x ;
faceinfo->yeyer= finder->tracker->faces[number]->eyeR_y ;
```

Fichier `DiskFile.cpp` :

- L'include au fichier fcntl.h a été enlevé.
- fmode = _O_BINARY a été enlevé
- L'appel à la fonction rewind(file) a été enlevé

Code Original:
Lignes 1 à 3 :

```
#include "DiskFile.h"
#ifdef _MSC_VER
#include <fcntl.h>
```

Modifications apportées :
Lignes 1 à 3 :

```
#include "DiskFile.h"
#ifdef _MSC_VER
#include <fcntl.h>
```

Code Original:
Lignes 13 à 14 :

```
     fmode = _O_BINARY;
#endif
```

Modifications apportées :
Lignes 13 à 14

```
13,16c13,14
//   _fmode = _O_BINARY;
#endif
```

Code Original:
Ligne 121 :

```
     ::rewind(file);
```

Modifications apportées :
Ligne 123

```
//   ::rewind(file);
```

Fichier `FaceTrackerAPI-MCTMSMFMV-0.cpp` :

- "rb" au lieu de "r" (binary mode)

Code Original:
Ligne 86 :

```
DiskFile *ifile = new DiskFile(strConcat(&str_allocated,
dir_name_, "pyramid-cascade.model"), "r");
```

Modifications apportées :
Ligne 86 :

```
DiskFile *ifile = new DiskFile(strConcat(&str_allocated,
dir_name_, "pyramid-cascade.model"), "rb");
```

Fichier `FaceTrackerAPI-MCTMSMFMV-1.cpp` :

Même remarque que précédemment :
- "rb" au lieu de "r" (binary mode)

Code Original:
Ligne 56 :

```
DiskFile *ifile = new DiskFile(strConcat(&str_allocated,
dir_name_, "pyramid-cascade.model"), "r");
```

Modifications apportées :
Ligne 56 :

```
DiskFile *ifile = new DiskFile(strConcat(&str_allocated,
dir_name_, "pyramid-cascade.model"), "rb");
```

Fichier `ipSWmct.cpp` :

Même remarque que précédemment :
- "rb" au lieu de "r" (binary mode)

Code Original:
Ligne 22:

```
file = new DiskFile(model_filename_, "r");
```

Modifications apportées :
Ligne 22 :

```
file = new DiskFile(model_filename_, "rb");
```

Fichier `ipSWmctMS.cpp` :

Même remarque que précédemment :
- "rb" au lieu de "r" (binary mode)

Code Original:
Ligne 23:
```
file = new DiskFile(model_filename_, "r");
```

Modifications apportées :
Ligne 23 :
```
file = new DiskFile(model_filename_, "rb");
```

Fichier `ipSWmvfaceBtree-mctMS.cpp` :

Même remarque que précédemment :
- "rb" au lieu de "r" (binary mode)

Code Original:
Ligne 81:
```
file_ = new
DiskFile(strConcat(&str_allocated[n_str_allocated++],
dir_name_, inplane_models_filename_), "r");
```

Modifications apportées :
Ligne 81 :
```
file_ = new
DiskFile(strConcat(&str_allocated[n_str_allocated++],
dir_name_, inplane_models_filename_), "rb");
```

Code Original:
Ligne 204:
```
file_ = new
DiskFile(strConcat(&str_allocated[n_str_allocated++],
dir_name_, outplane_models_filename_), "r");
```

Modifications apportées :
Ligne 204 :
```
file_ = new
DiskFile(strConcat(&str_allocated[n_str_allocated++],
dir_name_, outplane_models_filename_), "rb");
```

Code Original:

Ligne 315:

```
file_ = new
DiskFile(strConcat(&str_allocated[n_str_allocated++],
dir_name_, inplane_connect_models_filename_), "r");
```

Modifications apportées :
Ligne 315 :

```
file_ = new
DiskFile(strConcat(&str_allocated[n_str_allocated++],
dir_name_, inplane_connect_models_filename_), "rb");
```

Code Original:
Ligne 366:

```
file_ = new
DiskFile(strConcat(&str_allocated[n_str_allocated++],
dir_name_, outplane_connect_models_filename_), "r");
```

Modifications apportées :
Ligne 366 :

```
file_ = new
DiskFile(strConcat(&str_allocated[n_str_allocated++],
dir_name_, outplane_connect_models_filename_), "rb");
```

Annexe : Parties de codes sur la detection de visage avec la nouvelle librairie (explications des fichiers)

```
/*****************************************************************************************************/
/*** Function track( char* worldModelPath,                                                         ***/
/***                 char* fileNameToTrack,                                                        ***/
/***                 char grayMode,                                                                ***/
/***                 _ff_faceinfo_t** faceinfo,                                                    ***/
/***                 float score)                                                                  ***/
/*****************************************************************************************************/
/*** This function is a DLL exported function                                                      ***/
/*** it tracks a face on a image using the worldmodelpath "frontal.model"                          ***/
/*** and the image is loaded with the local function ppm_loadbmp                                   ***/
/***                                                                                              ***/
/*** 5 parameter  : 1) the path to the face detecton model File "frontal.model"                    ***/
/***                2) the path to the bmp file with a face to track                              ***/
/***                3) grayscaling mode selection in case multiple grayscaling functions are      ***/
/***                   implemented                                                                ***/
/***                4) pointer to the face infomation structure in order to set it                ***/
/***                   (x,y)face coordinates, (x,y)left eye & right eye coordinates               ***/
/***                5) the score gives the reliability of the face tracking                       ***/
/***                   in case of authentication                                                  ***/
/*** return value : BOOL(specifing if a face was tracked)                                         ***/
/*****************************************************************************************************/
extern "C" __declspec(dllexport)BOOL __stdcall track(char* worldModelPath,
                                                     char* fileNameToTrack,
                                                     char grayMode,
                                                     _ff_faceinfo_t** faceinfo,
                                                     float score)
{
    //local variable declarations specifing face & faced detection information
    int *values;
    float myscore;
    int nb_faces;
    int current;

    //locale variable specifing image information
    unsigned char *image = NULL;
    int image_width;
    int image_height;
    int tmp_image_width;
    int tmp_image_height;
    int current_image;
```

Nous chargeons l'image en vérifiant que nous avons une image RGB 24 bits valide

```
//FaceFindingAPI_0.2.2 variable declaration
ff_finder_t finder;
ff_faceinfo_t ffaceinfo;
ff_image_type image_type;

//call to local function to load bmp file
image_type = ppm_loadbmp(fileNameToTrack,&image_width,&image_height,&image);

if (image_type == -1)
{
    /* unsupported image */
    exit (1);
}

if (!image)
{
    /* problem loading the image */
    exit (1);
}

/*"It creates the initial setup to perform face detection.
    Parameters:
    nb_max_faces    maximum detected faces in an image
    image_width     image width in number of pixels
    image_height    image height in number of pixels
    mode            operating mode: detection (single image) or tracking (video)
    view            view mode: frontal or multiview
    models_path     directory containing the face detection model" (FaceFindingAPI_0.2.2 doc)*/
finder = ff_finder_create( 1,
                           image_width,
                           image_height,
                           FF_MODE_DETECT,
                           FF_VIEW_FRONTAL,
                           worldModelPath);
```

Configuration du détecteur de visage avec spécifications sur l'image et sur le dossier contenant le model à appliqué (frontal.model)

```
    /*"change the value of a floating point setting (warning: you must call ff_settings_apply ()
    to apply the new settings to the finder)"(FaceFindingAPI_0.2.2 doc)*/

    //scanning with different factors
    ff_settings_set_f (finder, FF_SETTINGS_F_SCALE_FACTOR, 0.2);
    ff_settings_set_f (finder, FF_SETTINGS_F_STEP_Y_FACTOR, 0.2);
    ff_settings_set_f (finder, FF_SETTINGS_F_STEP_X_FACTOR , 0.2);
    ff_settings_set_i(finder,FF_SETTINGS_I_AVERAGE_LENGTH,2);
    //we specify the maximum and minimum faces' size
    ff_settings_set_i(finder,FF_SETTINGS_I_FACE_SIZE_MAX,(int)image_width-30);
    ff_settings_set_i(finder,FF_SETTINGS_I_FACE_SIZE_MIN,(int)image_width/3);

    //applying the previous settings
    ff_settings_apply (finder);

    /*" It processes the image and returns the number of detected faces.

        Parameters:
        finder  the face detection finder
        image   the image to process
        type    the image type: Gray or Rgb24

        Returns:
        the number of detected faces "      (FaceFindingAPI_0.2.
    nb_faces = ff_process(finder,image,image_type);
    printf("nr of faces%i\n",nb_faces);

    //Face information is returned here
    ffaceinfo = ff_faceinfo_get(finder,0);

    //setting the reference parameter with the face information
    *faceinfo = ffaceinfo;

    //free memory using IDIAP's FaceFindingAPI 0.2.2 functions
    ff_finder_release(finder);
    ff_faceinfo_release(ffaceinfo);

    //free memory
    free(image);

    //to be sure it went through the function
    return true;
}
```

Paramètres d'optimisation du détecteur qui permet de choisir plusieurs paramètres dont la grandeur d'un visage, le scanning de l'image (choix entre qualité de la détection ou la rapidité de la détection)

Détection de visage et récupération des informations

Nous libérons la mémoire

Annexe : Portage de « l'ancienne » libraire de l'IDIAP (Torch3Vision)

Ce chapitre regroupe toutes les modifications apportées au code fournis par l'IDIAP en montrant à chaque fois le code original et les modifications apportées

Modification des fichiers d'en-tête (Header Files)

Fichier `DiskXFile.h` :

Retrait du mot clé « static »

Code Original:
Ligne 15 :

```
static bool is_native_mode;
```

Modifications apportées :
Ligne 17 :

```
/*static*/ bool is_native_mode;
```

Code Original:
Lignes 34 à 40 :

```
static bool isLittleEndianProcessor();
/// Returns #true# if the processor uses the big endian coding format.
static bool isBigEndianProcessor();

/// Returns #true# if we'll load/save using the native mode.
static bool isNativeMode();
```

Modifications apportées :
Lignes 36 à 42 :

```
/*static*/ bool isLittleEndianProcessor();
/// Returns #true# if the processor uses the big endian coding format.
/*static*/ bool isBigEndianProcessor();

/// Returns #true# if we'll load/save using the native mode.
/*static*/ bool isNativeMode();
```

Code Original:
Ligne 46 :

```
static void setNativeMode();
```

Modifications apportées :
Ligne 48:

```
/*static*/ void setNativeMode();
```

Code Original:
Ligne 52 :

```
static void setLittleEndianMode();
```

Modifications apportées :
Ligne 54:

```
/*static*/ void setLittleEndianMode();
```

Code Original:
Ligne 58 :

```
static void setBigEndianMode();
```

Modifications apportées :
Ligne 60:

```
/*static*/ void setBigEndianMode();
```

Modification des fichiers sources (Source Files) :

Fichier `DiskXFile.cc` :

Code Original:
Lignes 1 à 4 :

```
#include "DiskXFile.h"
#include "string_utils.h"
#ifdef _MSC_VER
#include <fcntl.h>
```

Modifications apportées :
Lignes 1 à 6:

```
#include "DiskXFile.h"
#include "string_utils.h"
#include <windows.h>
#ifdef _MSC_VER
//#include <fcntl.h>
#include <stdlib.h>
```

Code Original:
Ligne 9 :

```
bool DiskXFile::is_native_mode = true;
```

Modifications apportées :
Ligne 11:

```
//bool DiskXFile::is_native_mode = true;
```

Code Original:
Lignes 14 à 19 :

```
_fmode = _O_BINARY;
#endif

its_a_pipe = false;
bool zipped_file = false;
if(!strcmp(open_flags, "r"))
```

Modifications apportées :
Lignes 16 à 22:

```
//_fmode = _O_BINARY;
#endif

is_native_mode = true;
its_a_pipe = false;
bool zipped_file = false;
if(!strcmp(open_flags, "rb"))
```

Code Original:
Ligne 34 :

```
file = fopen(file_name, "r");
```

Modifications apportées :
Lignes 37 à 45:

```
int count = 0;
file = fopen(file_name, "rb");
while (!file && count < 10){
        count ++;
        Sleep(1);
        file = fopen(file_name, "rb");
}
```

Code Original:
Lignes 39 à 41 :

```
file = popen(cmd_buffer, open_flags);
if(!file)
  error("DiskXFile: cannot execute the command <%s>", file_name, cmd_buffer);
```

Modifications apportées :
Lignes 50 à 52:

```
//  file = popen(cmd_buffer, open_flags);
if(!file)
  error("DiskXFile: cannot execute the command <%s>", file_name, cmd_buffer);
```

Code Original:
Lignes 48 à 50 :

```
file = fopen(file_name, open_flags);
if(!file)
  error("DiskXFile: cannot open <%s> in mode <%s>. Sorry.", file_name, open_flags);
```

Modifications apportées :
Lignes 59 à 69:

```
int count = 0;
file = fopen(file_name, open_flags );

while (!file && count < 10){
	count ++;
	Sleep(1);
	file = fopen(file_name, open_flags);

}
//if(!file)
//	error("DiskXFile: cannot open <%s> in mode <%s>. Sorry.", file_name, open_flags);
```

Modifications apportées :
Ajout d'une ligne supplémentaire ligne 62 dans le code original ou ligne 81 sur le code modifié

```
is_native_mode = true;
```

Modifications apportées :
Ajout de code supplémentaire dans la fonction read() pour le debbugage:
Lignes 94 à 97:

```
if(melanie!= n_blocks)
{
	message("problem");
}
```

Code Original:
Ligne 115 :

```
::rewind(file);
```

Modifications apportées :
Lignes 140 à 141:

```
fseek( file , 0L , SEEK_SET );
//::rewind(file);
```

Code Original:

Lignes 220 à 223 :

```
if(its_a_pipe)
  pclose(file);
else
  fclose(file);
```

Modifications apportées :
Lignes 246 à 249:

```
if(!its_a_pipe)
   fclose(file);
//else
  //pclose(file);
```

Fichier `ipSWmct.cc` :

- "rb" au lieu de "r" (binary mode)

Code Original:
Lignes 22 :

```
file = new(allocator) DiskXFile(model_filename_, "r");
```

Modifications apportées :
Lignes 22

```
file = new(allocator) DiskXFile(model_filename_, "rb");
```