ENSICAEN
6, bd maréchal Juin
F-14050 Caen cedex
4

Spécialité Informatique
1re année

ENSICAEN

Rapport de mini projet

Création de vidéos avec ffmpeg

Griffon <griffon@ecole.ensicaen.fr>
Blanchard <blanchar@ecole.ensicaen.fr>
Elouarti <ouarti@ecole.ensicaen.fr>

Suivi ENSICAEN : JEHAN-BESSON Stéphanie

2e semestre 2005-2006

2e semestre 2005-2006

Table des matières

Introduction

Il existe, de nos jours, de nombreux formats vidéo propriétaires ou libres. Aux vues de leur diversité, il est intéressant de disposer d'outils tels que ffmpeg et ffdshow capables manipuler, analyser et convertir ces vidéos assez simplement. FFmpeg est un logiciel open source permettant de convertir rapidement des fichiers vidéo et audio qui est sans cesse en évolution, ce qui lui permet de reconnaître les formats les plus récents. FFdshow permet quand à lui d'analyser de manière plus efficace les videos en étudiant leur flux de données.

Dans de nombreux domaines tels que les trucages au cinéma, on a parfois besoin de naviguer dans les images. Nous avons ainsi élaboré un logiciel qui permettant la navigation dans une image en utilisant une fonctionnalité de ffmpeg. Celui-ci pourra permettre lavalidation des outils d'estimation de mouvements développés au laboratoire de recherche GREYC.

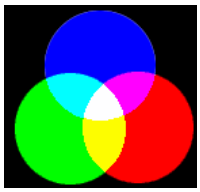
1 Introduction sur les formats vidéo

1.1 Espaces couleur

La couleur tel que nous l'entendons peut se représenter selon deux critères simples :
– La chromaticité : c'est-à-dire la teinte globale de la couleur.
– L'intensité lumineuse : la quantité de lumière contenue dans la couleur, c'est l'aspect clair ou foncé.
Lorsque l'on représente une image en couleur, on souhaite conserver ces deux informations, tandis que pour une image en noir et blanc (ou en niveaux de
gris), on ne garde que l'intensité lumineuse.
Avant tout, il faut choisir un modèle de représentation, en voici trois exemples :

Le système RGB (Red, Green, Blue) :

Le système RGB est le codage le plus simple et le plus répandu. Il s'appuie sur la synthèse additive adaptée pour la représentation sur l'écran. Chaque couleur est représentée par son niveau de rouge, de vert et de bleu.

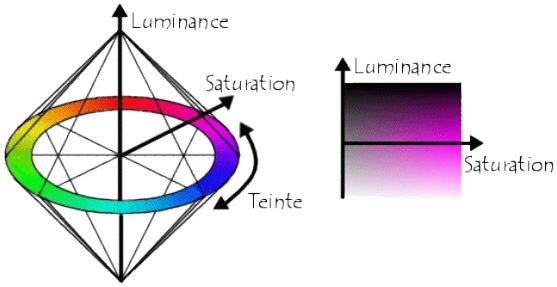


Synthèse additive des couleurs

Le modèle HSL (Hue, Saturation, Luminance) :

Le modèle HSL s'appuie sur une modélisation plus naturelle de la couleur et plus proche de la perception de l'oeil humain. C'est une alternative au modèle RGB qui ne facilite pas toujours le choix de la couleur.

On décompose ainsi la couleur en trois critères physiologiques :
- La teinte : Elle correspond `a la perception chromatique de la couleur. On peut la représenter par un cercle chromatique allant du rouge au violet. On passe ainsi par toutes les nuances intermédiaires : orange, jaune, vert, bleu.
- La saturation : Elle se rapporte à l'intensité ou pureté de la couleur. La saturation permet de distinguer les couleurs vives, pastels ou délavées.
Augmenter la saturation consiste à rajouter un pourcentage de gris dans la couleur.
- La luminosité : Elle indique la quantité de lumière de la couleur, la luminosité indique si une couleur est sombre ou claire. En augmentant la luminosité on se rapproche du blanc, en la diminuant, on se rapproche du noir.



Représentation du modèle HSL

Le modèle YUV :
Le modèle YUV définit un espace colorimétrique en trois composantes. Le premier représente la luminance et les deux autres représentent la chrominance .YUV est utilisé dans les systèmes de diffusion télévisuelle et il est considéré comme étant le modèle qui se rapproche beaucoup plus de la perception humaine des couleurs que le standard RGB utilisé dans l'imagerie informatique.

Dans le traitement des signaux vidéo couleur, on ajuste les coefficients des signaux U et V de telle sorte que la valeur crête à crête soit égale à la valeur maximale de Y, soit 0,7 V.

Cette relation peut être utilisée pour dériver *Y*, *U* et *V* à partir des composantes *R*, *G* et *B* :

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

1.2 Formats vidéo

1.2.1 Le format AVI

L'**Audio Vidéo Interleave** (audio vidéo entrelacée) (dont l'acronyme est **AVI**), est un format de fichier conçu pour stocker des données audio et vidéo.

AVI utilise un même paquet standard afin d'être lu simultanément.

Ce paquet standard est un format de fichier permettant de rassembler en un seul fichier :

- un ou des flux *vidéo*
- un ou des flux *audio*
- d'autres données, par exemple :
    - descriptions des flux que contient le conteneur
    - des méta données (auteur, date, etc.)
    - des sous-titres
    - chapitrages

Les flux audio et vidéo sont compressés ou décompressés à l'aide de codecs.

Les principaux conteneurs vidéo sont :

- NUT;
- OGM (plus évolués) ;
- Matroska (plus évolués) ;
- AVI

- Quick time
- ASF.

Dans un fichier AVI, chaque composante audio ou vidéo peut être compressé par n'importe quel codecs. Le format DivX est souvent utilisé comme codecs vidéo, et le format mp3 comme codecs audio, mais d'autres codecs peuvent également être utilisés, par exemple XviD ou MPEG pour la vidéo, et mp2, WAV etc. pour l'audio.

Le format AVI permet de réunir en un seul fichier une piste vidéo et jusqu'à 99 pistes audio, ce qui permet de bénéficier, par exemple, de plusieurs langues pour un même film.

1.2.2 Les formats MPEG

Les réunions du *Moving Picture Experts Group* ont démarré en 1988 dans le but de développer un premier standard, MPEG-1, pour des applications de stockage audio/vidéo du type Vidéo CD. MPEG a ensuite rapidement produit un second standard, MPEG-2, visant essentiellement les applications liées à la télévision numérique. D'autres familles de standards ont depuis été produites.

MPEG a développé les standards suivants :

- **MPEG-1** : leur premier standard audio et vidéo utilisé plus tard comme standard des Vidéo CDS. Ce format offre une résolution à l'écran de 352 x 240 pixels à 30 images par seconde ou de 352 x 288 à 25 images par seconde avec un débit d'environ 1,5 Mbit/s.Il inclut le populaire format audio MPEG-1 Layer 3 (MP3).
- **MPEG-2** : standard couvrant le codage de l'audio et la vidéo, ainsi que leur transport pour la télévision numérique, et (avec quelques restrictions) pour les vidéo-disques DVD ou SVCD. C'est notamment le format utilisé jusqu'à présent pour la TV sur ADSL. Les débits habituels sont de 2 à 6 Mbit/s pour la résolution standard (SD), et de 15 à 20 Mbit/s pour la haute résolution (HD)
- **MPEG-4** : norme comblant le vide des bas débits (jusqu'à 2 Mbit/s) pour lesquels MPEG-2 n'avait pas été développé. Il permet entre autres de coder des objets vidéo/audio, le contenu 3D et supporte le DRM. La partie 2 de MPEG-4 (Visuel) est compatible avec la partie baseline de H.263 et a connu du succès grâce à l'implémentation DivX ainsi que dans les téléphones mobiles. La partie 10 appelée **MPEG-4 AVC** permet des gains d'un facteur 2 à 3 par rapport à MPEG-2 et a déjà été retenu comme le successeur de celui-ci pour la TV haute définition, la TV sur ADSL et la TNT.

H.264, ou **MPEG-4 AVC**, est une norme de codage vidéo qui permet de créer une nouvelle architecture de codecs ayant pour but un gain en efficacité de codage d'un rapport au moins égal à 2 par rapport aux standards existants (**MPEG-2**, **H.263** et **MPEG-4** Part 2).

Cette norme permet le codage de vidéo avec images progressives et entrelacées, éventuellement combinées dans une même séquence.

Elle permet aussi de créer une interface simple pour pouvoir adapter le codec aux différents protocoles de transport (commutation de paquets et de circuits). Le codec a été développé en s'assurant qu'il serait implémentable sur plate-forme à un coût raisonnable, c'est à dire en tenant compte des progrès réalisés par l'industrie des semi-conducteurs en matière de design et le procédés.

La prédiction compensée de mouvement :

La prédiction compensée de mouvement est un puissant moyen pour réduire les redondances temporelles entre trames et elle est utilisée dans MPEG-1 et MPEG-2. Le concept de la compensation du mouvement est basé sur l'estimation du mouvement entre trames vidéo; si tous les éléments d'une scène vidéo sont proches dans l'espace, le mouvement entre trames peut être décrit avec un nombre limité de paramètres (vecteurs de mouvement des pixels).

La meilleure prédiction d'un pixel est donnée par la prédiction de mouvement de la trame précédente. Bien que, soit la prédiction de l'erreur que les vecteurs de mouvement sont transmis, le codage de l'information de mouvement pour chaque pixel de l'image n'est pas nécessaire.

Si la corrélation spatiale entre vecteurs de mouvement est assez haute, un vecteur de mouvement pourra représenter un bloc de pixels adjacents.

Ces blocs sont souvent constitués d'un groupe de 16x16 pixels (comme lors de la compression jpeg), et seulement un vecteur de mouvement est estimé, codé et transmis pour chaque bloc.

Ces vecteurs peuvent être visionnés grâce à des logiciels tels que ffdshow.

1.3 Format raw

RAW (de l'anglais raw, qui signifie brut) est un format d'images numériques. Ce n'est pas un standard, mais plutôt une désignation commune pour les fichiers générés par les dispositifs tels qu'appareils photos numériques, ou scanners, et n'ayant subi aucun traitement. Ces fichiers sont généralement spécifiques à chaque constructeur d'appareil, voire à chaque appareil, et nécessitent donc un logiciel spécial pour pouvoir les exploiter.

Un fichier au format RAW est un fichier numérique peu compressé qui contient les informations brutes enregistrées par le capteur de l'appareil photo. Ce fichier est en quelque sorte « en attente de développement » dans la mesure où il n'a subi aucun des traitements de linéarisation, contraste, luminosité ou saturation, nécessaires pour produire une image lisible.

Comme le format mpeg, le format raw usuel utilisé en vidéo se base sur l'espace couleur YUV.

2 Quelques fonctionnalités de ffmpeg

2.1 Présentation de ffmpeg

FFmpeg est un logiciel open source qui peut enregistrer, lire ou convertir un flux numérique, audio ou vidéo. Il est développé sous Linux, mais il peut être compilé sur la plupart des systèmes d'exploitation, notamment windows. Le projet ffmpeg est hébergé par SourceForge (ffmpeg.sourceforge.net/). Il était téléchargeable sous Linux grâce à CVS et maintenant avec SVN.

2.1 Utilisation de ffmpeg

2.2.1 Conversion de fichiers vidéo et audio

FFmeg permet de convertir les formats de fichiers vidéo et audio. Il décode et encode un très grand nombre de formats. On retrouve notamment les formats vidéos avi, les différents mpeg, Quicktime ou YUV, les formats audio wav et mp3, mais aussi des formats d'image comme ppm, jpeg ou gif.

La commande pour convertir est intuitive: L'option '-i' permet de spécifier le fichier source. Cet exemple convertit un avi en mpg:

```
ffmpeg -i test.avi out.mpg
```

Il est également possible de générer de la même manière des gif animés (non compressés) à partir d'une vidéo :

```
ffmpeg -i test.avi out.gif
```

De même pour les fichiers audio: ici un wav est convertit en mp3 à 22050 Hz:

```
ffmpeg -i test.wav -ar 22050 out.mp3
```

2.2.2Combinaison entre fichiers audio et fichiers vidéo

Sur le même modèle que la conversion de fichier, il est possible de fusionner un fichier audio et un fichier vidéo pour créer une vidéo avec une piste sonore:

```
ffmpeg -i son.wav -i video.yuv out.mpg
```

A l'inverse, on peut aussi extraire le flux audio d'un fichier video:

```
ffmpeg -i test.avi out.mp3
```

2.2.3 Création de vidéos à partir d'un ensemble d'images

Il est également possible de générer une vidéo à partir d'un ensemble d'images: Les images sont diffusées l'une après l'autre suivant le framerate souhaité (option '-r'), avec une compression fixée (option '-b'). Puis on donne les fichiers sources et le fichier de sortie.

```
ffmpeg -r 24 -b 1800 -i %02d.bmp out.mpg
```

Dans l'exemple ci-dessus, l'enchaînement est de 24 images par seconde, avec un taux de compression 1800 Kbits par seconde. Les images sources choisies sont successivement 00.bmp, 01.bmp, 02.bmp,... selon le nombre d'images existantes.

2.3 Quelques options disponibles de ffmpeg

`-L'
Voir la licence
`-h'
Voir l'aide
`-formats'
Voir les formats, codecs et protocoles disponibles
`-i nomFichier'
Fichier d'entrée
`-y'
Ecrasement du fichier de sortie
`-title chaineDeCaracteres'
Definir le titre de la vidéo .
`-author chaineDeCaracteres'
Définir le nom de l'auteur.
`-copyright chaineDeCaracteres'
Définir le copyright.
`-comment chaineDeCaracteres'
Inclure des commentaires.
`-target type'
Spécifier le format du fichier sortant
`-b bitrate'
Spécifier le taux de compression en kb/s(par défaut = 200 kb/s).
`-r fps'
Spécifier le nombre d'images par seconde (par défaut = 25).
`-s size'
Spécifier la resolution de la vidéo.
Le format est `Largeurxhauteur' (par défaut = 160x128).
Les abbreviations suivantes sont reconnues:
`sqcif'
128x96
`qcif'
176x144
`cif'
352x288
`4cif'
704x576

`-aspect aspect'
Définir l’aspect (4:3, 16:9 ou 1.3333, 1.7777).
`-maxrate bitrate'
Spécifier la tolérance maximale du taux de compression (en kbit/s).
`-minrate bitrate'
Spécifier la tolérance minimale du taux de compression (en kbit/s).

2.4 Présentation de ffdshow et des fonctionnalités disponibles

ffdshow est un puissant pack de codecs permettant sous Windows de décoder et encoder des fichiers audio et vidéos dans les formats les plus répandus. Pour cela, ffdshow utilise des bibliothèques de ffmpeg. De plus ffdshow propose un nombre considérable d'options (pour sa version récente). Après installation, ffdshow se lance automatiquement et simultanément au lecteur (par exemple Windows Media Player) lors de l'ouverture d'une vidéo. Un icône apparaît alors dans la zone de notification de la barre des taches et permet d'accéder à la configuration et aux options de ffdshow. On peut ainsi modifier les paramètres vidéo ou bien audio (égaliseur, convolution, filtrage...) mais dans le cadre de notre projet, seul le traitement de l'image nous intéresse.

Les fonctionnalités de ffdshows sont les suivantes:
_ Choix pour chaque format vidéo d'utiliser ffdshow ou non et de choisir la bibliothèque à utiliser pour décoder si plusieurs gèrent ce format;
_ Affichage d'informations incrustées dans la vidéo lue. On peut par exemple connaître les propriétés des images (durée, numéro, taille), la durée de la vidéo ou le temps déjà écoulé, le nom de la source, le débit, le codec utilisé pour la lecture ou encore l'espace couleur de la sortie;

Les options suivantes permettent quant à elles de modifier l'apparence de la vidéo (Certaines d'entre-elles peuvent s'appliquer à l'image entière ou seulement à sa moitié droite pour comparer le résultat du traitement à l'image d'origine):

_ Zoom et recadrage pour les vidéos dont les bords sont altérés ou pour donner un aspect 16:9 à une vidéo 4:3 dont les parties inférieures et supérieures ne sont pas intéressantes à voir;
_ Désentrelacement par une douzaine de méthodes différentes;
_ Filtrage;
_ Post-traitement;
_ Réglage du contraste, de la luminosité, de la teinte, de la saturation, correction gamma;

_ Niveaux;

_ Décalage spatial de luminance et chrominance;

_ Flou et réduction du bruit;

_ Accentuation: effet de netteté, mais assez peu détectable;

_ Profondeur de chant: donne à l'image un effet "peinture à l'huile";

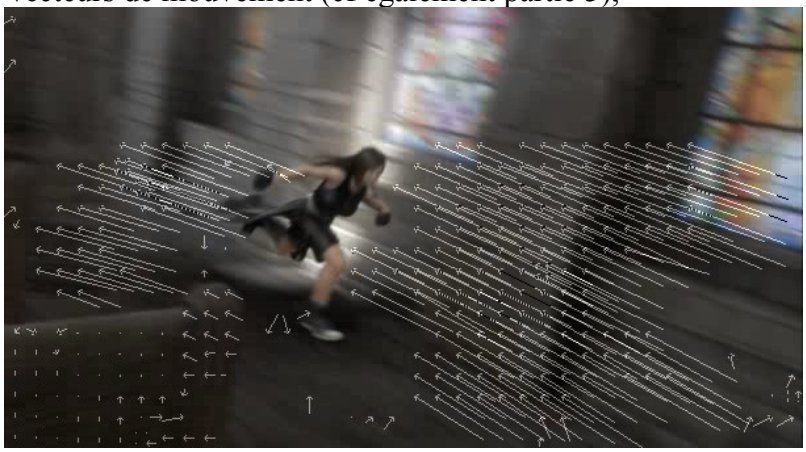
_ Ajout de bruit (grain, tremblement spatial, tremblement d'intensité, ajout de poussières, de rayures);

_ Redimensionnement: pour corriger les vidéos dont le ratio d'origine n'est pas respecté;

_ Correction des perspectives;

_ Gestion des sous-titres: affichage, correction orthographique (peu de langues disponible, non testé), modification de la police et de la couleur;

_ Affichage des vecteurs de mouvement (cf également partie 3);



Pour mouvement proche d'une translation

Pour une rotation couplée à un zoom

_ Filtrage DCT;

_ ajout d'une image fixe par plusieurs méthode (assombrissement, éclairsissement, mélange, éclusion, ajout);

_ Capture d'image (une seule ou une séquence) en jpg, bmp ou png;
_ Réglage de sortie: retournement vertical de l'image, choix de l'espace couleur (par défaut, ffdshow choisit automatiquement pour chaque vidéo l'espace couleur qui semble le mieux adapté);

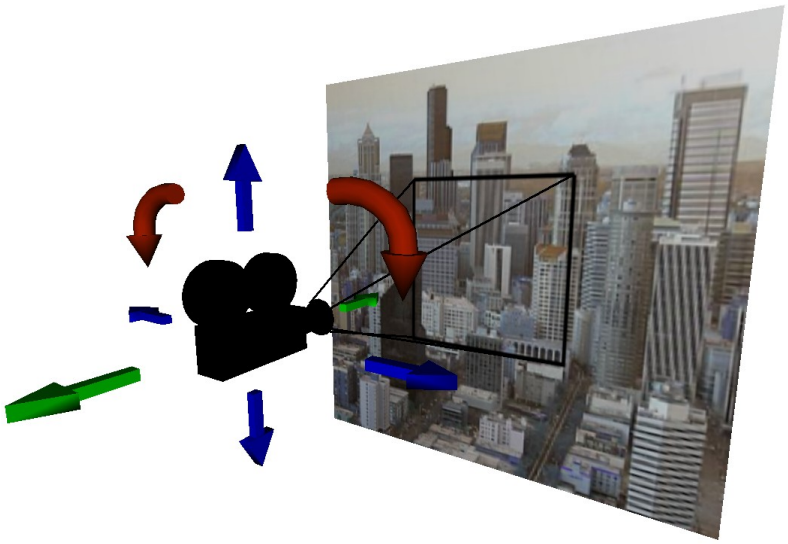
Ffdshow permet donc de visionner un grand nombre de vidéos de formats différents, mais permet aussi de modifier l'aspect des vidéos pour corriger les imperfections ou appliquer des effets particuliers à travers un important panel d'options.

3 Notre programme de génération de mouvement de caméra : ffcam

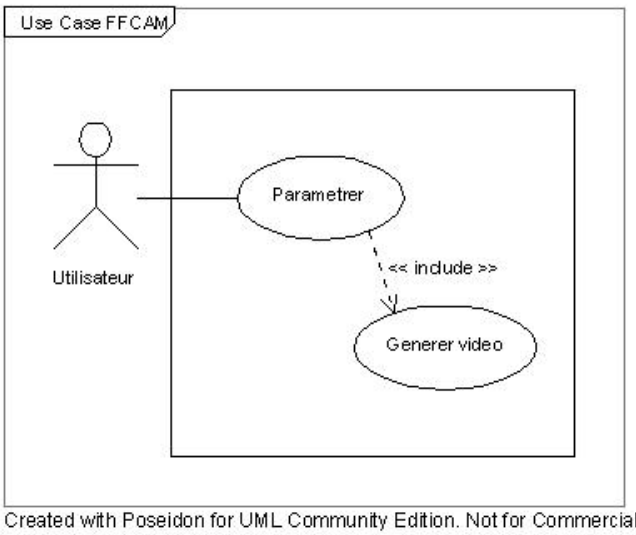
3.1 Structure du projet

3.1.1 Objectifs

L'objectif du programme est de générer des mouvements de caméra à partir d'une grande image de base au format ppm.
Les mouvements de caméra peuvent se faire dans les plans de l'image (mouvement de translation) mais aussi sur un axe perpendiculaire à ce plan (effet de zoom sur l'image) et enfin la caméra peut effectuer une rotation autour de cet axe.



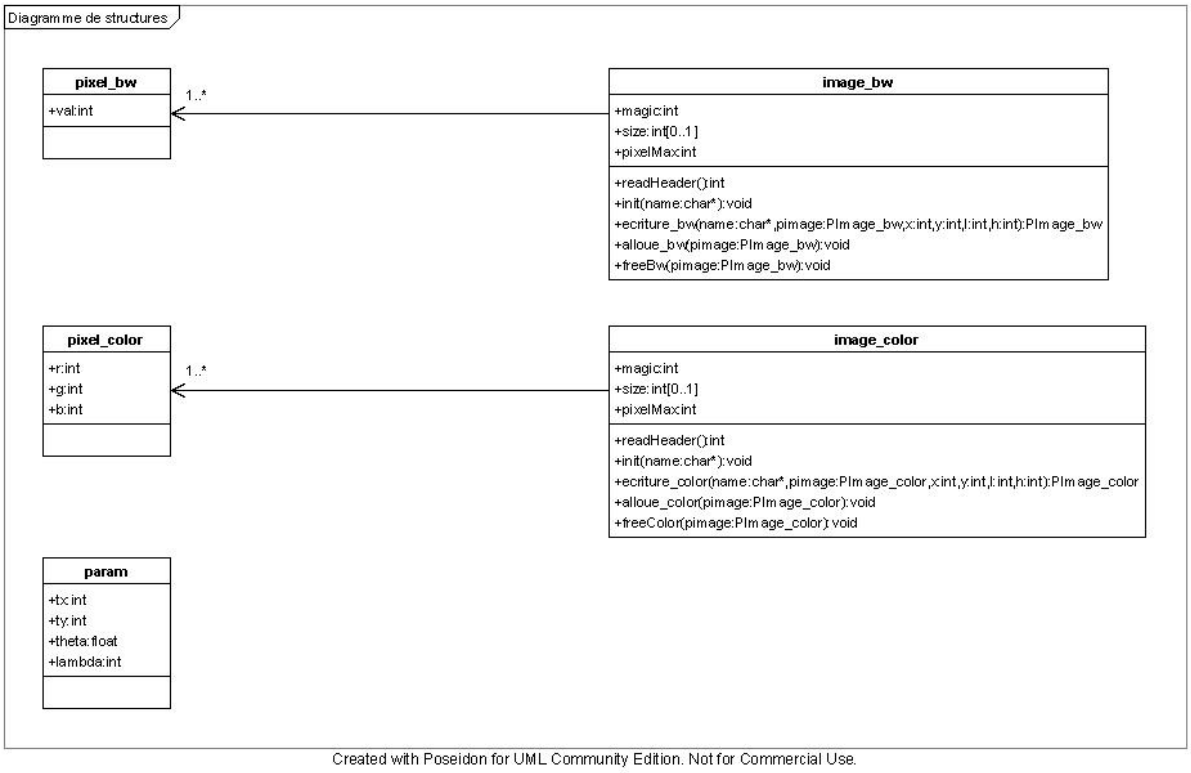
L'utilisation du programme se doit d'être assez simple. L'utilisateur paramètre le mouvement qu'il souhaite appliquer à l'image et le programme gère lui-même la génération de la vidéo.



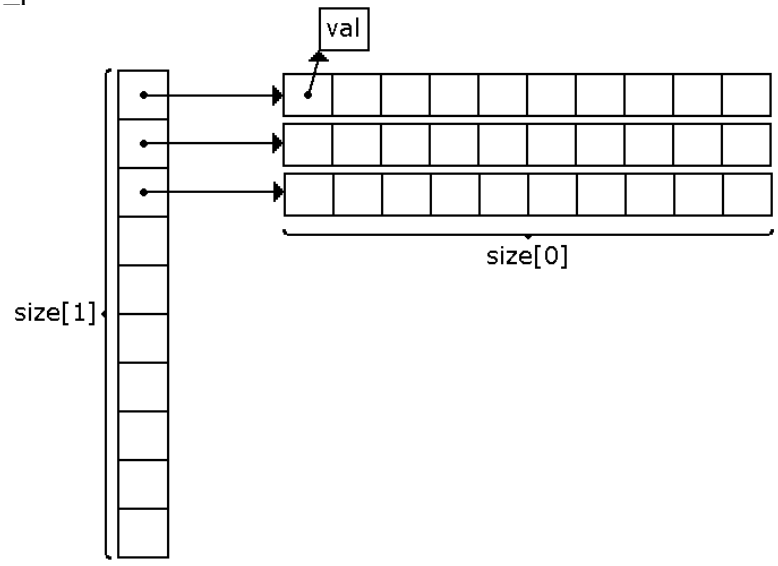
3.1.2 Structures et arborescence

Une bibliothèque dynamique de gestion de ppm a été générée, car elle pourra être utilisée ultérieurement par d'autres programmes (cf. Tgenerator en annexe 4.4). Elle permet de charger une image ppm dans une structure image_bw ou image_color selon le type de fichier, et aussi de sauvegarder une partie de l'image chargée dans un nouveau fichier.

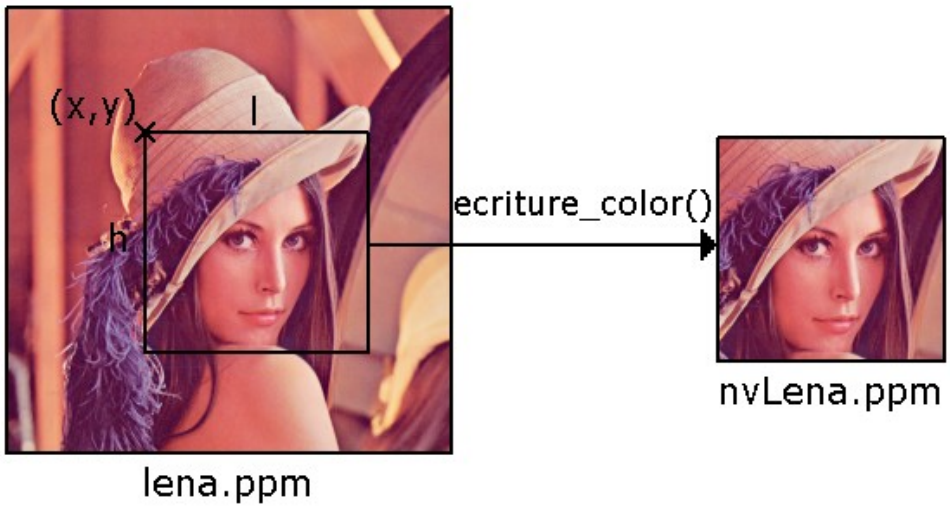
Voici les différents types de structure utilisés dans le programme (cf annexe 4.2):



L'initialisation d'une image passera par l'allocation de mémoire nécessaire pour stocker les données dans tab_pixel. Elle se fait ainsi :

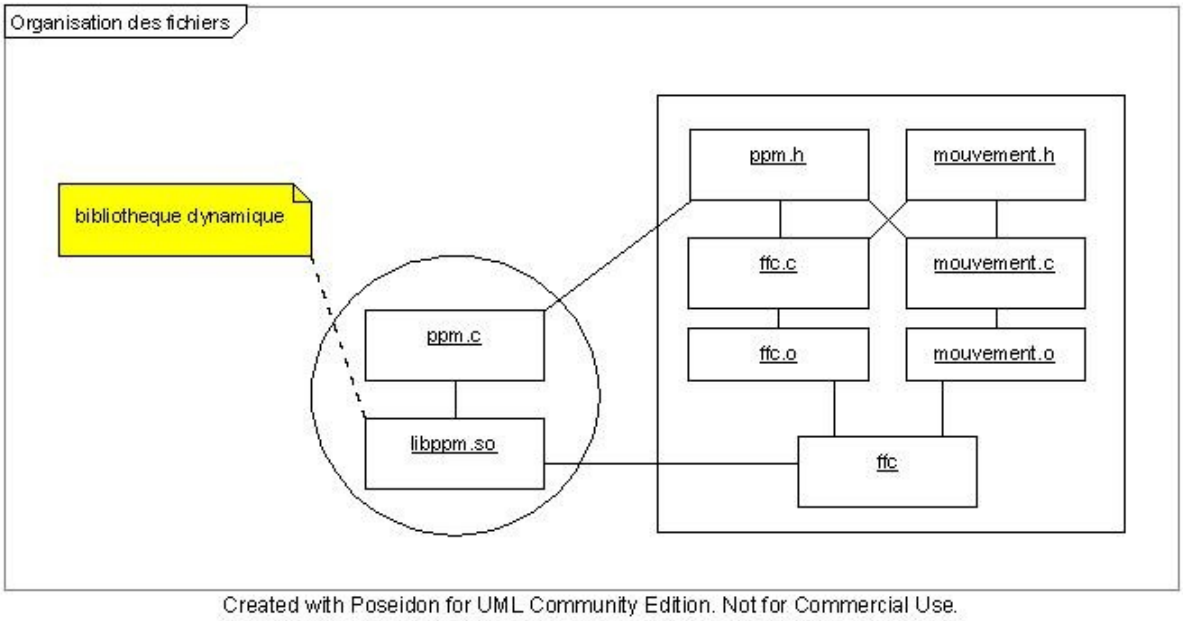


Les fonctions ecriture_bw() et ecriture_color permettent de sauvegarder une partie de l'image (cf annexe 4.2):



Le fichier mouvement.c contient les fonctions qui permettent de manipuler l'image afin de simuler un mouvement de caméra. Nous expliquerons les algorithmes utilisés dans ces fonctions par la suite.
Le fichier central est ffc.c, il lit les paramètres du mouvement à simuler dans un fichier texte passé en argument et appelle en boucle les fonctions de mouvement.c afin de créer une suite d'images qui, en appelant ffmpeg, donneront la vidéo désirée.
Un script, ffcam, permet de coordonner l'appel du programme final ffc et celui de ffmpeg.

L'organisation des fichiers est donc :



3.1.3 Manuel d'utilisation

Installation

Copiez le contenu de l'archive sur votre disque. Puis dans le dossier ffcam/src/, tapez la commande make install.

Utilisation

Dans le dossier ffcam, vous pouvez appeler le programme ffcam avec les paramètres souhaités. Voici la syntaxe:

./ffcam [image.ppm] [move.txt] [fps] [bitrate] [video]

image.ppm est le nom du fichier image source, move.txt le nom du fichier contenant les instructions de mouvement de camera, fps le nombre d'images par seconde, bitrate le taux de compression de la video et enfin video le nom de la vidéo créée.

Création d'un fichier de vecteurs mouvement

Il existe en fait deux types de fichiers de vecteurs compatibles avec ffcam. Le premier permet d'effectuer des mouvements quelconques, association d'une translation, d'une rotation et d'un zoom. Le second type de fichier se limite au simple mouvement de translation, mais il permet un calcul plus rapide de la vidéo et peut être généré automatiquement avec l'outil Tgenerator (cf annexe 4.4) à partir d'une image où est dessinée la trajectoire à suivre.

Fichier de type 1 (mouvements complexes):

La première ligne est composée de quatre entiers séparés par un espace. Les deux premiers entiers ne sont pas utilisés. Les deux suivant correspondent à la résolution de la vidéo (longueur puis hauteur en pixels).

La seconde ligne comprends la lettre 'm' qui signifie que le fichier est de type 1 (mouvements complexes), puis du nombre de mouvements de la séquence. Dans l'exemple, on effectue quatre mouvements avant d'en rajouter un cinquième dans une deuxième séquence. Il est à noter que la séparation de la vidéo en séquences n'a aucune influence sur le rendu final, mais permet juste de séparer les différentes phases du mouvement lorsqu'on regarde ce fichier texte.

Les quatre lignes suivantes sont donc les quatre mouvements qui composent la séquence. Ils sont écrits dans l'ordre de leur réalisation et sont composés de quatre paramètres séparés par un espace: Les deux premiers paramètres sont le vecteur de translation décomposé suivant les axes x et y (l'origine d'une image est le coin supérieur gauche). Lors du premier mouvement, ces deux valeurs indiquent les coordonnées de départ de la vidéo. Viennent ensuite l'angle de rotation (entier en degré) et le facteur de zoom (réel).

Exemple 1:

```
0 0 320 240
m 4
100 50 0 1
2 4 0 1
0 0 -10 1
0 0 0 1.2
m 1
-2 -4 10 0.83
```

Remarque: Si la caméra sort en partie ou totalement de l'image, elle verra un fond noir.

Fichier de type 2 (translations seules):

La première ligne est composée de quatre entiers séparés par un espace. Les deux premiers entiers indiquent les coordonnées de départ. Les deux suivants correspondent à la résolution de la vidéo (longueur puis hauteur en pixels).

La seconde ligne comprend la lettre 't' qui signifie que le fichier est de type 2 (translations seules), puis du nombre de translations de la séquence. Dans l'exemple, on en effectue quatre avant d'en rajouter un cinquième dans une deuxième séquence.

Les quatre lignes suivantes sont donc les quatre translations. Les deux paramètres sont les coordonnées du vecteur de translation suivant les axes x et y (l'origine d'une image est le coin supérieur gauche).

Exemple 2:

```
100 50 320 240
t 3
1 1
2 2
2 -2
t 1
-2 -4
```

Remarque: Avec ce type, la vidéo est générée plus rapidement, mais si la caméra sort en partie de l'image, le programme est interrompu et la vidéo s'arrête à cet instant.

3.2 Description des algorithmes

3.2.1 Trois mouvements combinés: Translation, Rotation, Zoom

Lorsque l'on veut appliquer à une matrice de pixels divers mouvements sur ses valeurs, on applique généralement la formule qui suit :

$$\begin{bmatrix} x \\ y \end{bmatrix} = \lambda \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} xp \\ yp \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix}$$

Le vecteur [x y] représente les coordonnées du point considéré dans l'image de base, et [xp yp] les coordonnées du point correspondant à [x y] dans l'image ayant subit l'opération .
Le vecteur [tx ty] est un vecteur de translation, le scalaire λ effectue une homothétie sur le vecteur d'origine et donc un effet de zoom, enfin, la matrice en cosinus et sinus est une matrice de rotation d'angle θ.
Notre programme aurait pu ainsi appliquer cette formule à chaque pixel de la matrice d'origine pour ensuite prendre la partie d'image qui nous intéresse.
Cependant cette méthode possède quelques inconvénients car les coordonnées de pixels obtenues ne sont plus entières et nécessitent donc une approximation pour pouvoir être utilisées, approximation qui donnera sans doute, au final, une image ayant des pixels qui n'auront reçu aucune valeur.
Il est possible grâce à des méthodes d'interpolation de palier à ces problèmes. Cependant, ces algorithmes sont assez complexes et peuvent amener des résultats mitigés.

Nous avons donc opté pour une solution moins complexe et donnant malgré tout des résultats très corrects.
En effet, nous possédons, à la base, une image de taille importante mais où la caméra ne se focalise que sur une partie réduite. L'astuce sera donc d'appliquer la formule inverse sur une matrice de pixel n'ayant pour taille que la partie réduite de l'image. Pour chaque pixel de cette matrice, on cherchera la valeur d'un pixel de la grande image qui doit lui être attribuée.
L'avantage de cette technique est d'obtenir une matrice à laquelle chaque pixel a reçu une valeur (si le mouvement déborde de l'image, le pixel sera noir). De plus cette technique est moins gourmande puisqu'elle ne nécessite pas de traitement par interpolation.

Voici donc la formule qui est appliquée :

$$\begin{bmatrix} xp \\ yp \end{bmatrix} = \frac{\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} tx \\ ty \end{bmatrix} \right)}{\lambda}$$

Cependant appliquer cette formule directement ne donne pas le résultat escompté car ici, le point fixe est l'origine de l'image, en haut à gauche. Il faut donc avant d'effectuer ces opérations, translater ce point fixe.

L'opération se fait en deux étapes :

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} \frac{1}{2}l - tx \\ \frac{1}{2}l - ty \end{bmatrix}}{\lambda} + \begin{bmatrix} \frac{1}{2}l - tx \\ \frac{1}{2}l - ty \end{bmatrix}$$

Pour la translation /zoom.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} \frac{1}{2}l - tx \\ \frac{1}{2}l - ty \end{bmatrix} \right) + \begin{bmatrix} \frac{1}{2}l - tx \\ \frac{1}{2}l - ty \end{bmatrix}$$

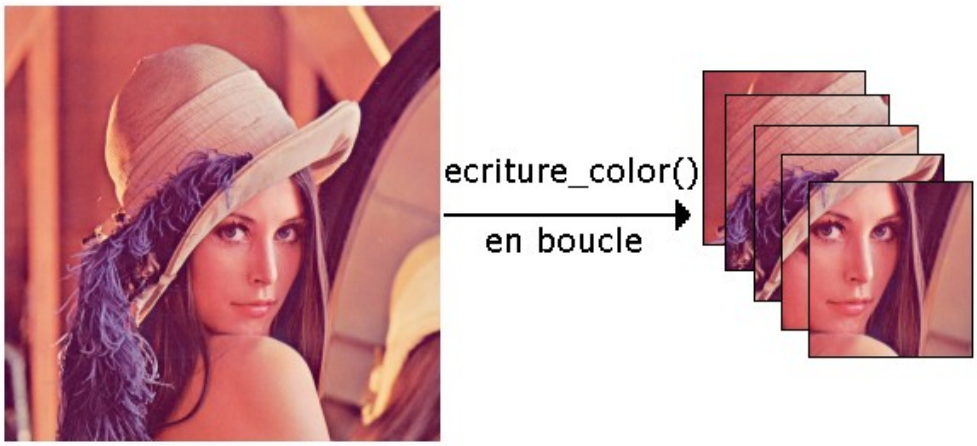
Pour la rotation.

Dans ces formules, h représente la hauteur de l'image voulue et l la largeur (en pixel).

3.2.2 Mouvement de translation optimisé

L'algorithme résultant des formules précédentes est assez rapide car il ne dépend que du nombre de pixels de l'image à créer et du temps de calcul d'un cosinus et d'un sinus. Mais il faut par la suite appeler la fonction écriture qui elle aussi est en O(n).

Vu que la fonction ecriture() de la bibliothèque libppm permet de sauvegarder un morceau de l'image quelconque, elle peut être utilisée pour simuler un effet de translation. En effet, il suffit de l'appeler en boucle en ne changeant que les coordonnées du pixel d'origine comme l'illustre ce schéma :



Cette astuce permet d'économiser l'appel des fonctions de mouvement lorsqu'elles ne sont pas nécessaires (pour une translation simple).

3.3 Exemples d'utilisations et comparaison des performances

3.3.1 Exemple de mouvements complexes

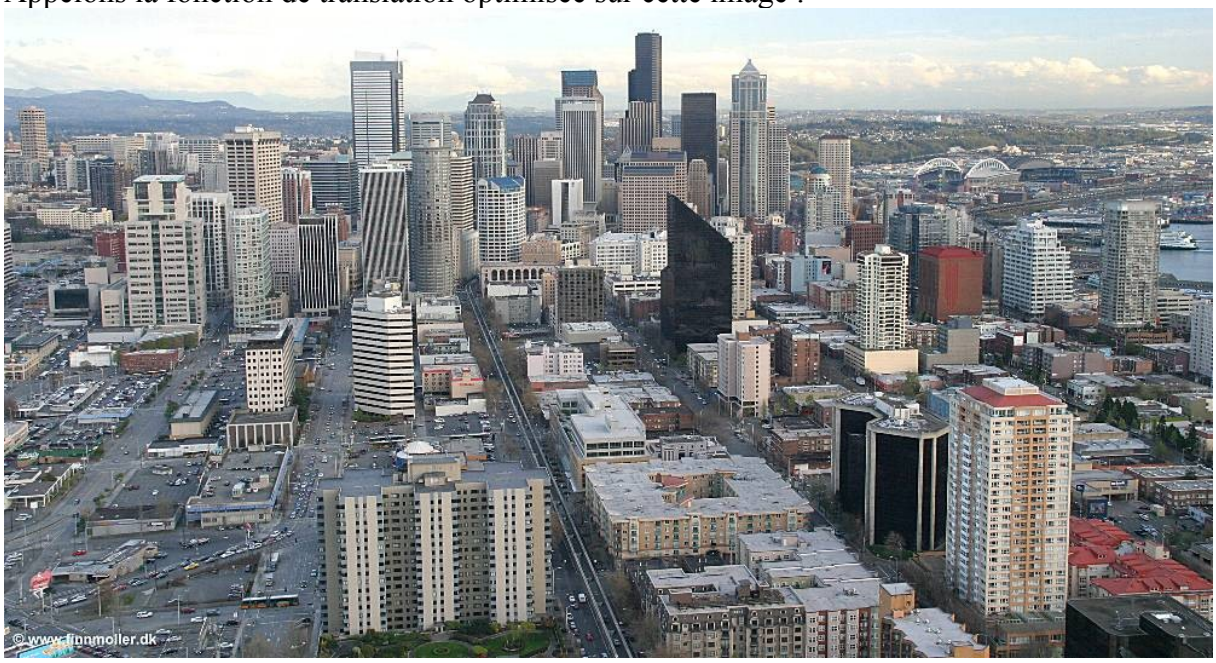
Prenons une image ayant une résolution de 1141x631 pixels :

Nous allons effectuer un mouvement combinant translation, zoom et rotation sur un cadre de 200x200 pixels au centre du village.
Les paramètres passés dans le fichier de vecteur sont 10 10 1 1.05 et sont répétés 24 fois pour ainsi obtenir une vidéo d'une seconde à 24img/s.
Il y aura ainsi 24 translations de vecteur [10 10] combinées avec une rotation d'1 degré et un zoom d'amplitude 1.05.

La vidéo obtenue s'apparente à ceci :

3.3.2 Exemple de translation

Appelons la fonction de translation optimisée sur cette image :



Enchaînons 3 translations de vecteur [20 20] sur un cadre de l'image de 200x200 pixels situé aux coordonnées [200 200] , le résultat obtenu est :

3.3.3 Comparaison des performances

Nous avons inclus dans le code la possibilité de mesurer le temps d'exécution du programme. Nous allons donc tester celui-ci avec un mouvement de translation en faisant appel aux deux fonctions possibles.

Pour effectuer 264 translations sur une image de 200x200pixels :

Le programme classique a mis 4.43 secondes pour générer les images,
Le programme de génération de translations amélioré a mis 1.8 secondes

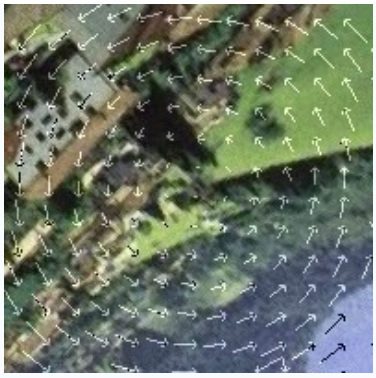
Pour effectuer 408 translations sur une image de 200x200pixels :

Le programme classique a mis 6.83 secondes pour générer les images,
Le programme de génération de translations amélioré a mis 2.79 secondes

On s'aperçoit donc bien que le programme optimisé est beaucoup plus rapide. L'écart grandissant en même temps que le nombre de sollicitations des fonctions, il doit donc y avoir au moins un degré de complexité de différence entre les deux algorithmes.

3.4 Visualisation des résultats avec ffdshow

En appliquant la fonctionnalité de visualisation de vecteurs de mouvement de ffdshow, on vérifie la nature du mouvement effectué.

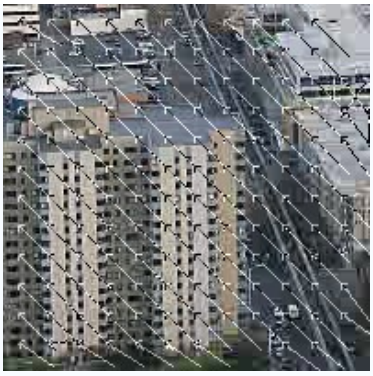


Translation

Sur une translation en diagonale vers le bas, les vecteurs sont tous de direction semblable, celle de le diagonale, et dirigés dans le sens inverse du mouvement, ce qui est le mouvement naturel des blocs de pixels auquel on aurait pu penser au préalable.

Rotation

Rotation de la caméra dans le sens horaire, les vecteurs sont dirigés dans le sens opposé du mouvement.



Zoom

Enfin pour un zoom d'amplitude 0.95 (s'éloignant de l'image) les vecteurs sont tous dirigés vers le centre de l'image.

Au final : notre programme couplé à ffdshow met en évidence la pertinence de la compression mpeg. La plupart des blocs de pixels résultant de la compression jpeg étant conservés d'une image à une autre, il est donc intéressant de ne stocker que leur déplacement dans l'image.

Conclusion

Nous avons donc vu la diversité des formats vidéo en étudiant les plus répandus (AVI, MPEG, YUV) et en constatant leurs différences.

Nous avons par la suite étudié les fonctionnalités les plus importantes de ffmpeg et ffdshow en constatant les facilités qu'ils apportent dans la manipulation de fichiers vidéo tels que ceux que nous avons pu créer avec notre programme ffcam.

Ce dernier bien qu'effectif, peut encore être optimisé dans son temps d'exécution en allouant une image dans un tableau unique, ce qui permettra par la suite une recopie plus rapide grâce à memcpy().

Références bibliographiques

[1] http://ffmpeg.mplayerhq.hu/ , documentation sur ffmpeg.

[2] Cours d'analyse vidéo, de Stéphanie Jehan-Besson.

[3] http://fr.wikipedia.org/wiki/MPEG , description de la compression mpeg

[4] http://fr.wikipedia.org/wiki/RGB , description de l'espace couleur RGB

[5] http://fr.wikipedia.org/wiki/teinte_saturation_lumière , espace HSL

Annexes: le code source

Annexe 1: listing

- ffcam		Script shell
- Tgenerator		Script shell
+ include		<rep>
|	ppm.h	Fichier d'en-tête
|	mouvement.h	Fichier d'en-tête
+ scr		<rep>
|	Makefile	Makefile
|	ppm.c	Fichier source c
|	mouvement.c	Fichier source c
|	ffc.c	Fichier source c
|	Tgenerator.c	Fichier source c