studio*de
(digital effects)

rapport de projet

fabien castan

guillaume maucomble

elica zaharieva

tuteur: venceslas biri

université de Marne-la-Vallée

imac ingenieur

REMERCIEMENTS

La réalisation de ce projet n'aurait sans doute pas été possible dans l'aide et le soutien de nos professeurs de manière générale. Nous tenons à adresser plus particulièrement nos remerciements à :

- Venceslas Biri
- Pascal Chaudeyrac
- Olivier Derpierre
- François de Sorbier de Pougnadoresse
- Jocelyne Kiss, pour avoir cru en notre projet
- l'ensemble des développeurs de la communauté GStreamer (et à sa mailing-liste très active)
- au site QtFr[1] pour leur aide toujours très rapide et très précise

1. *http://forum.qtfr.org/*

TABLE DES MATIERES

INTRODUCTION **6**

CONCEPTION THEORIQUE & RECHERCHES **7**

REFLEXION AUTOUR DES GRANDS PRINCIPES POUR LE DEVELOPPEMENT 8
CONCEPTION INTERFACIQUE 9
A) L'espace de l'écran et principes graphiques *9*
B) Parcours de l'utilisateur et l'expérience d'apprentissage *9*
C) Les types de données *10*
D) Bibliothèque Qt *11*
MORPHING (OU MORPHAGE EN FRANÇAIS) 13
A) Principe *13*
B) Le morphing entre deux images *14*
C) Le morphing entre deux vidéos *15*
Algorithmique 16
Amélioration 18
Implémentation 19
D) Logiciels existants *19*
DETECTION DES POINTS DE HARRIS 21
A) Principe *21*
B) Algorithmique *21*
C) Implémentation *23*
LE TRACKING 24
A) Les différentes approches dans le suivi de mouvement *24*
B) Les algorithmes de parcours *24*
C) Etude comparative des algorithmes de parcours *27*
D) Implémentations et tests des algorithmes de parcours *29*
E) Algorithmes de comparaison *31*
F) Résultats des tests des algorithmes de comparaison *32*
G) Implémentation finale *33*
H) Gestion d'erreur *34*

CONCEPTION TECHNIQUE & FONCTIONNALITES **36**

LA VIDEO 36
A) Présentation *36*
B) Le choix de GStreamer *36*
PLUGINS 39
A) Principe *39*
B) Algorithmique *39*
C) Implémentation *39*
INTERFACE GENERALE 41
A) Médias *41*
B) Moniteur *42*
C) Panneau d'effets *42*
D) TimeLine *42*
E) Menus *43*
INTERFACE DU PLUGIN MORPHING 46
A) Fonctionnement *46*
B) Prévisualisation *46*
C) Ajout / Suppression de points *47*

D) Création automatique *47*
E) Réglages du tracking *47*
F) Interface de la gestion d'erreur du tracking *48*

ANALYSE TECHNIQUE & ARCHITECTURE LOGICIELLE 49

LES CLASSES 51
A) Les données de projet *53*
B) L'application et la fenêtre principale *54*
C) Les classes du tracking *55*
D) Les classes du morphing *56*
E) Organisation des plugins *57*
EXECUTION DU PROGRAMME ET POINTS PARTICULIERS 58
A) Un motif de conception particulier : MVC *58*
B) Les autres motifs de conception *59*
C) Démarrage du programme *60*
D) Événements et drag' n' drop *61*
E) Structure de la sauvegarde *62*
ORGANISATION DES FICHIERS 63
CODAGE 67

RESULTATS 68

ORDONNANCEMENT DES TACHES 72

DROIT 73

A) Librairies *73*
B) Projet *73*

PROMOTION 74

A) Charte graphique *74*
B) Site Internet *75*

BILAN 77

BIBLIOGRAPHIE & REFERENCES 78

A) Générale *78*
B) Morphing *78*
C) Points de Harris *78*
D) Tracking *78*
E) Plugins *79*
F) Vidéo *79*
G) Qt & Interface *79*
H) Droit *80*

ANNEXES 81

A) Cahier des charges commanditaire *81*
B) Carnet de bord des réunions *81*
C) Guide d'utilisation *81*

INTRODUCTION

Le présent dossier est le résultat d'une année de travail autour du projet Studio*DE (Digital Effects), dans le cadre du programme universitaire de la seconde année de formation au diplôme d'ingénieur I.M.A.C. Il présente tous les aspects de notre produit final.

Dans le Cahier des Charges Commanditaire de Décembre 2006 (joint en annexe au présent document), nous avions défini les éléments contextuels du projet. L'étude menée nous avait alors permis de constater que notre produit pouvait présenter un intérêt dans le monde du logiciel libre, notamment sous Linux puisque ce système fournit peu de logiciels pour le traitement vidéo. Nous avions alors proposé le sujet consistant en la création d'une interface de montage permettant d'importer un média vidéo et de lui appliquer un traitement de morphing. Pour ce faire, un outil de tracking vidéo a dû être développé.

La première partie, concernant la conception théorique, fait principalement état de nos recherches, tant sur le point algorithmique (méthodes de tracking et morphing) que d'interfaçage avec l'utilisateur. C'est ici que nous décrivons les choix que nous avons dû prendre. La partie conception technique nous permet alors de présenter en détail les fonctionnalités du logiciel final.

A ce document nous joignons une analyse technique détaillant les clusters composant l'architecture informatique. Nous y détaillons l'organisation des ressources du point de vue du développeur.

Nous pourrons alors établir un bilan sur les résultats obtenus et présenter ainsi l'application résultante de notre travail.

Il nous faudra alors présenter les autres aspects de notre projet, à savoir le traitement de la question du droit et de la promotion de l'application. Ces deux points sont en effets non négligeables lors de l'élaboration d'un projet d'envergure.

Nous fournissons enfin le guide d'utilisation de l'application, disponible au téléchargement sur le site officiel du projet studiode.free.fr.

Ce dossier conclut la phase de réalisation de notre projet. Il permettra au lecteur de suivre toutes les étapes de réflexion qui nous ont menés au rendu final. Le bilan fourni en fin nous permettra d'émettre nos impressions et nos critiques sur le projet, sur l'organisation mise en place pour le réaliser.

Dans l'optique d'offrir aux utilisateurs de logiciels libres un produit permettant de réaliser un montage vidéo non linéaire prenant en charge de nombreux formats vidéo et offrant une évolutivité sans limite, nous avons pris le pari de mettre en place la base d'une application dédiée : Studio*DE (Digital Effects). Elle apporte une gamme d'effets graphiques et éditoriaux créatifs et de qualité directement au montage.

CONCEPTION THEORIQUE & RECHERCHES

Notre projet consiste en une application de montage créatif qui propose aux utilisateurs une alternative aux logiciels payants ou trop compliqués à prendre en main, comme cela peut être le cas actuellement sous Linux. Divisé en deux parties, le projet s'articule d'une part autour de la conception d'une interface de montage, et d'autre part autour d'un effet transitionnel entre deux médias, à savoir un morphing. Pour assister la gestion du mouvement dans une vidéo, le morphing utilise un module de tracking optimisé. Notre cible est à priori assez large. Une légère connaissance du montage non linéaire est certes recommandée, mais le guide d'utilisation doit permettre à un novice d'appréhender le logiciel et de découvrir toutes ces fonctionnalités. Le projet se veut également évolutif et participatif, c'est-à-dire que des développeurs externes devraient facilement pouvoir développer des modules directement intégrables à l'application, par exemple l'ajout d'effets.

Le cahier des charges commanditaire dessinait notre ambition et le concept initial. Il s'agit maintenant de préparer de manière définitive la phase de réalisation. Le cahier des charges techniques est là pour nous permettre de définir exactement le programme, sa manière de fonctionner. Nous avons donc pour cela étudié les solutions techniques qui nous permettent de réaliser notre projet.

Ainsi la phase de réalisation est préparée en amont grâce à ce document. Nous savons d'ores et déjà ce que nous voulons réaliser. Définis dans la cahier des charges commanditaire, les diagrammes de Pert et Gantt nous ont permis d'une part d'estimer l'envergure du projet, en détaillant les tâches de manière pourtant succincte, d'autre part d'approximer des délais de recherches et réalisation, bien que ceux-ci soient clairement soumis à modifications. Le présent dossier contient d'ailleurs des plannings « réalistes » tenant compte de la réalité des contraintes temporelles et matérielles qui nous sont imposées.

Dans le dossier précédent nous avions présumé l'utilisateur comme un utilisateur de système Linux désirant manipuler un logiciel d'édition vidéo modulaire et complet, qui pourrait éventuellement participer à l'amélioration du logiciel si ses capacités de développement le lui permettent. N'oublions pas que la Linux et la communauté du logiciel libre en général tendent à s'élargir et de plus en plus de personnes sont intéressées par les outils qu'ils offrent, par les fonctionnalités qu'ils proposent et qui concurrencent les grands logiciels, et la gratuité de ceux-ci.

L'intérêt est également double pour nous. Le premier est de proposer une solution alternative intéressante, et le second d'offrir à cette solution la possibilité d'être étendue très rapidement grâce aux développeurs intéressés par le projet.

Le cahier des charges commanditaire proposait une série de fonctionnalités[2] que nous souhaitions traiter. Au terme des cinq mois de conception et réalisation depuis la parution de ce

2. *Voir « V.2 Les grands principes de fonctionnement », Cahier des charges commanditaire, Studio*DE(Digital Effects), 2006*

dossier, nous avons pu recentrer notre approche et nos attentes. Dans la partie ci-après nous établissons une analyse et critique des points que nous avions évoqués, pour ensuite s'appliquer à examiner les points que nous avons réellement traités, à savoir l'interface, le tracking et le morphing. Nous y ferons l'état de nos recherches, nos choix et des critiques qui y sont associées. Nous pourrons ensuite présenter les fonctionnalités implémentées pour l'application dans la conception avancée.

REFLEXION AUTOUR DES GRANDS PRINCIPES POUR LE DEVELOPPEMENT

Dû aux contraintes techniques, temporelles et humaines qui nous étaient imposées, nous avons établis plusieurs heuristiques. Elles avaient été avancées dans le cahier des charges et une rapide étude nous a permis de les clarifier.

Le développement général de l'application et de ses modules se fait avec le langage de programmation C++. Forts d'un semestre d'apprentissage de ce langage, nous avons trouvé beaucoup d'avantages à l'utiliser, notamment le fait qu'il autorise la programmation orientée objet, fondée sur le concept d'objet (association de données et de méthodes traitant ces données au sein d'une même entité, ce dernier point révélant le principe d'encapsulation), qui permet des développements intéressants et puissants.

L'application permet d'ajouter des fonctionnalités supplémentaires en ajoutant simplement des fichiers « plugins » dans un répertoire associé. Nous détaillerons ce point dans la présentation de la conception avancée.

Nous avions évoqué à l'époque la possibilité de développer une application de montage total grâce à laquelle on disposerait de fonctionnalités originales, notamment une gestion de calques vidéos qui lierait le concept de calques en logiciel d'édition photo, et le concept de couches en logiciels d'édition vidéo. Conscients alors des difficultés du développement notamment du point de vue des ressources à disposition (3 personnes pour s'occuper du projet, de nombreux points à traiter en priorité), nous avons dû nous séparer de cette ambition. Cependant, elle pourra être implémentée facilement plus tard.

Nous voulions également faire un logiciel multilingue. Là encore, le temps imparti ne nous a pas suffi pour décliner la langue utilisée, cependant, l'utilisation de la bibliothèque Qt nous permet de ne pas nous inquiéter sur la faisabilité de ce point. En effet, l'application a été conçue dans un but de régionalisation, et il nous sera facile d'associer plusieurs fichiers de langage au programme.

CONCEPTION INTERFACIQUE

A) L'ESPACE DE L'ECRAN ET PRINCIPES GRAPHIQUES

La charte graphique de l'application doit être soignée. Notre cible étant large, il est indispensable que l'interface soit intuitive. Les signes et images devront être bien interprétés par l'utilisateur, donc mûrement réfléchis par nous. Il ne faut pas en effet imposer à l'utilisateur un apprentissage trop long de l'interface qui lui est proposée.

Dans un produit tel que le notre, l'utilisateur attend une utilisation ludique, il veut être constructeur de son processus créatif et être autonome. Nous proposerons donc une interface simple dans laquelle il pourra s'orienter sans recourir à une aide quelconque.

L'espace dont nous disposons est celui de l'écran. De rapport généralement 4:3, l'utilisateur peut afficher l'application en mode plein écran ou non. Nous définissons cependant une taille minimale obligatoire pour pouvoir accéder à toutes les fonctionnalités proposées sur l'interface.

B) PARCOURS DE L'UTILISATEUR ET L'EXPERIENCE D'APPRENTISSAGE

Un logiciel tel que le notre est caractérisé par la non-linéarité associée à son utilisation, autorisant des parcours de lecture et d'utilisation multiples. C'est le principe d'interactivité rendue possible car il agit sur son propre parcours d'utilisation.

Pour manipuler le logiciel, l'utilisateur doit passer par une première étape d'apprentissage pour se construire une représentation du schéma global de l'application en termes de fonctionnalités et limites. On pourrait déterminer cinq grandes dimensions, ou types d'activités, qui peuvent être inclus dans l'expérience d'apprentissage[3] et sur lesquelles nous nous basons pour réaliser une interface claire.

- **Activité représentationnelle**: Elle s'exerce sur des représentations, un objet (textes, images) et permet l'assimilation, la maîtrise et le traitement du contenu. Appréhender l'interface revient à acquérir la maîtrise des concepts et théories régissant son fonctionnement.
- **Activité opérative**: l'apprentissage ne peut pas se réduire à une construction de schémas de fonctionnement. Pour connaître les méthodes de fonctionnement, il faut manipuler, construire, essayer. Le but est de maîtriser par l'expérimentation et la manipulation les objets de l'environnement proposé. Elle mobilise alors

3. *Voir la thèse de Philippe Charlier, Hypermédias et expérience d'apprentissage, 2000, université de Louvain*

essentiellement les ressources de la perception et de la motricité. Ce point permet de caractériser l'interactivité de notre application.

- **Activité réflexive**: elle est liée à la distanciation et l'abstraction par rapport à l'activité. Elle implique un acte de création qui appartient à l'utilisateur. C'est ce qui caractérise sont autonomie, qui débouche ensuite vers un engagement dans l'action, c'est à dire vers le désir de manipuler l'application pour arriver à ce qui a été réfléchi. Notre application en elle-même ne nécessite pas de réel besoin de distanciation. Nous entendons par là que pour effectuer une opération sur un média ou sur son projet en général, l'utilisateur ne doit pas avoir besoin d'un temps de distanciation de l'application pour réfléchir. Par contre, pour son objet même (montage créatif), il est nécessaire à l'utilisateur de passer par cette étape de réflexion. Il ne faut donc pas que nous lui imposions une réflexion double, la première concernant son expression créative, la seconde l'utilisation du logiciel. Cette dernière doit être intuitive pour ne pas gâcher l'expérience de l'utilisateur.

- C'est le point soulevé par **l'activité psychoaffective**. Le plaisir d'utilisation est une dimension vitale à un projet tel que le notre (on peut même l'étendre à tous les projets informatiques dont la cible est très large). L'utilisateur manipule des vidéos pour le plaisir d'obtenir un résultat convaincant, nous devons lui fournir un outil capable de répondre à ses attentes sans frustrer son expérience d'utilisation.

- **L'activité sociale**. Cette dernière activité a pour objet l'interaction sociale. L'apprentissage s'inscrit dans un cadre culturel véhiculant des règles et des normes qui l'affectent directement. Ainsi notre application se doit d'être elle-même inscrite dans ce consensus. A ce titre, l'interface est proche de celles des autres logiciels d'édition vidéo, pour ne pas notamment dérouter ou perdre l'utilisateur.

Pour le projet Studio*DE, nous avons tenté d'exploiter ces dimensions au maximum et rendre ainsi l'utilisation confortable et intuitive.

La dimension sociale se retrouve plutôt (mise à part l'utilisation de conventions sociales de présentation des éléments) autour du concept de développement modulaire, et ne concerne alors pas tous les utilisateurs. Nous parlons ici du fait que les développeurs volontaires pourront s'ils le souhaitent ajouter leurs propres effets et les partager pour toute la communauté des utilisateurs. Enrichissant ainsi l'application, celle-ci se retrouve dotée de fonctionnalités puissantes et nombreuses, et s'impose ainsi face aux autres logiciels.

C) LES TYPES DE DONNEES

Nous devrons nous occuper de plusieurs types de données pour notre application. La première concernant évidemment les types de médias qu'on peut importer et exporter à partir de Studio*DE.

- **Médias vidéos**

  Les médias vidéo sont de deux types – soit un fichier vidéo, soit une suite de fichiers image. Le framework GStreamer à été retenu pour lire les fichiers vidéo. Malheureusement sa gestion au sein de notre application n'est pas encore opérationnelle. La structure est prévue, mais la lecture de la vidéo en soi, à partir de GStreamer, pose problème. Il faut donc pour l'instant travailler dans le programme avec des suites d'images. Cependant, à l'avenir, grâce à la librairie GStreamer nous pourrons gérer de très nombreux formats vidéo. GStreamer fonctionnant par plugin, l'utilisateur pourra installer les plugins qu'il souhaite pour gérer les formats dont il a besoin et les utiliser de façon transparente dans notre application.

- **Médias photos**

  Les formats d'images gérer sont les plus classiques : jpeg, jpg, ppm, pgm, tga. Cependant grâce à Qt nous pourrions rajouter la gestion de très nombreux formats facilement. Mais à l'avenir GStreamer devrait nous permettre également de gérer des images fixes.

- **Médias sonores**

  Actuellement l'application ne gère pas le son. Nous nous sommes en effet consacrés à la mise en œuvre d'un plugin de transition entre deux vidéos d'un point de vue image. Bien sûr GStreamer nous permettra à l'avenir de gérer l'audio très facilement.

D) Bibliothèque Qt

Une bibliothèque logicielle (ou librairie) est un regroupement de fonctions qui permettent de réaliser des tâches concernant le même domaine. On les utilise dans un logiciel pour se libérer de la programmation entière du programme. Pour les utiliser il faut donc disposer de leur code source ou d'une interface permettant d'accéder aux fonctions. Dans la conception initiale, nous avions plusieurs librairies graphiques pour la gestion de l'interface graphique, à savoir Qt, Juce, SDL, GTK ou encore wxWidget. Nous avons étudié les possibilités de chacun d'entre eux. La raison pour laquelle nous avons opté pour Qt (« cute ») est qu'elle offre en plus des composants graphiques des véritables fonctionnalités de codage (gestion de Threads pour la manipulation temporelle par exemple) non négligeables. De plus, c'est une librairie approchée au cours de notre formation, ce qui nous a permis de nous familiariser avec son mode de fonctionnement.

Nous avons précédemment démontré que l'environnement que nous proposons se doit d'être intuitif. Les contraintes évoquées nous posent le problème de l'originalité de l'interface. L'intégration 3D du logiciel nous aurait permis d'engager une réflexion sérieuse sur les nouveaux processus d'appréhension d'un logiciel de montage créatif. Cependant une réflexion sur nos attentes vis-à-vis de l'application et de ses fonctionnalités nous a montré qu'une interface communément admise car utilisée par la majorité des logiciels d'édition vidéo nous permettait de nous affranchir de l'étape d'apprentissage lié à celle-ci. Notre orientation aurait pu accompagner celle de Avid Pinnacle Studio qui propose une interface divisée en « étapes », ce qui permet de

disposer notamment de plus d'espace à l'écran et d'une utilisation assez simple à comprendre. Cependant, conscients des limites que cela implique (notamment pour le passage d'une étape à une autre, et le retour à une étape précédente), et conscients que notre cible n'avait pas forcément envie de cliquer trop souvent pour accéder à des outils spécifiques, nous nous sommes alignés sur le consensus qui rapproche l'interface de l'ordinateur aux bancs de montage analogiques (un écran de visualisation, une time lime qui s'impose comme la métaphore du banc de montage, des outils de manipulation des vidéos qui remplacent la manipulation physique des pellicules).

Qt nous permet ainsi de développer en C++ une application riche en outils de types boutons, éléments graphiques etc. La version utilisée est la 4.2.2. Il est conseillé d'utiliser celle-ci et seulement celle-ci pour exécuter le programme, car les versions de cette plateforme ont tendance à ne pas être totalement compatibles entre elles.

Dans la partie suivante (« *Conception technique*») nous présenterons les résultats de ces analyses en détaillant l'interface utilisateur de l'application.

MORPHING (OU MORPHAGE EN FRANÇAIS)

Le premier plugin de notre application de traitement vidéo est un effet de morphing. Cet effet est le premier effet numérique de l'histoire du cinéma. Il est utilisé pour la première fois dans le film « *Willow*» en 1988. Il fut tellement employé par la suite qu'il finit presque par devenir un effet trop connu et les grosses productions ont tendance à éviter son utilisation.

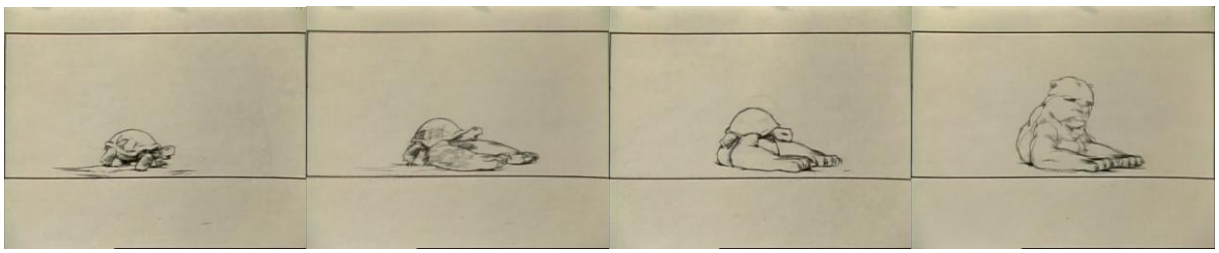

Fig. 1 : Story-board du film "Willow" (1988)

20 ans plus tard de nombreuses méthodes ont été crées. Les films utilisent maintenant souvent un morphing 3D. De nombreux petits logiciels du commerce proposent cet effet entre deux images, mais ils sont déjà beaucoup plus rares à le proposer entre deux vidéos. En vidéo, cela demande énormément de travail à l'utilisateur, car il faut faire des correspondances entre beaucoup d'images des deux vidéos. Nous proposons une méthode pour automatiser au maximum ce travail en se basant sur du tracking. Celle-ci à l'inconvénient d'avoir des cas particuliers (occlusions, entrées/sorties du cadre, etc.).

A) Principe

Le morphing est une interpolation géométrique et colorimétrique.

Fig. 2 : Morphing entre deux figures géométriques

L'idée du morphing est de faire une transformation entre 2 objets de façon réaliste. Le spectateur doit croire que le premier objet filmé se transforme en un nouvel objet. Le principe consiste donc à déformer les deux images l'une vers l'autre grâce à des points de correspondances entre les elles (cliqués par l'utilisateur).

Un autre effet proche est le warping qui consiste simplement à déformer une image. Le morphing revient donc à faire un fondu entre deux warpings.

Fig. 3 : Exemple de warping

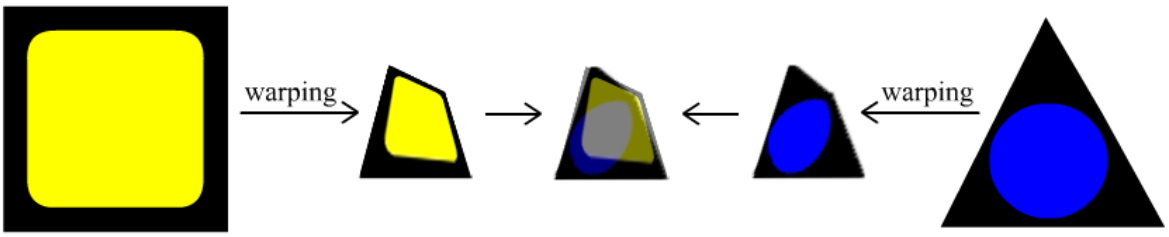

Fig. 4 : Le morphing est l'équivalent d'un fondu entre deux warpings

En réalité, on part de l'image résultat et en fonction de la transformation on va chercher le point correspondant dans les deux images sources (avec un mode d'interpolation que l'on choisit). Le pixel enregistré est le mélange des deux pixels trouvés avec un coefficient qui dépend de la position dans le temps par rapport à la durée du morphing.

B) LE MORPHING ENTRE DEUX IMAGES

Fig. 5 : Morphing entre deux images (réalisé avec Winmorph)

Il existe plusieurs techniques pour faire un morphing qui apporte des résultats très peu différents (plus ou moins précis), mais qui diffèrent avant tout par la façon de travailler de l'utilisateur qui voudra réaliser un morphing, et par le temps de calcul.

Les différentes méthodes se basent sur un maillage, sur des courbes (ou éventuellement d'autres formes géométriques) ou sur des points de correspondances. Nous avons choisi une méthode basée sur des points de correspondances pour permettre une solution automatisée (voir partie suivante « morphing vidéo ») et pour les temps de calcul. Pour réaliser cela, il reste deux solutions :

- calculer la position d'un pixel par rapport à tous les points de l'image

- calculer la position d'un pixel seulement par rapport à 3 points, c'est à dire la plus petite entité qui le contienne : le triangle

Nous avons opté pour la deuxième solution qui est beaucoup moins coûteuse car dans l'optique d'une utilisation entre deux vidéos, il nous faut minimiser les temps de calcul. Mais avec un ensemble de points, il est possible de réaliser une multitude de triangles différents. Pour contourner ce problème nous avons utilisé la triangulation de Delaunay qui, à partir d'un ensemble de points, réalisera une triangulation où les triangles seront le plus équilibrés possible, mais sans rajouter de points (contrairement à certains algorithmes qui imposent des contraintes sur les angles, etc...). On entend par « équilibré » le fait de limiter au maximum les triangles plats, qui lors du morphing créeraient de mauvaises déformations et pourraient faire apparaître les triangles.

C) LE MORPHING ENTRE DEUX VIDEOS

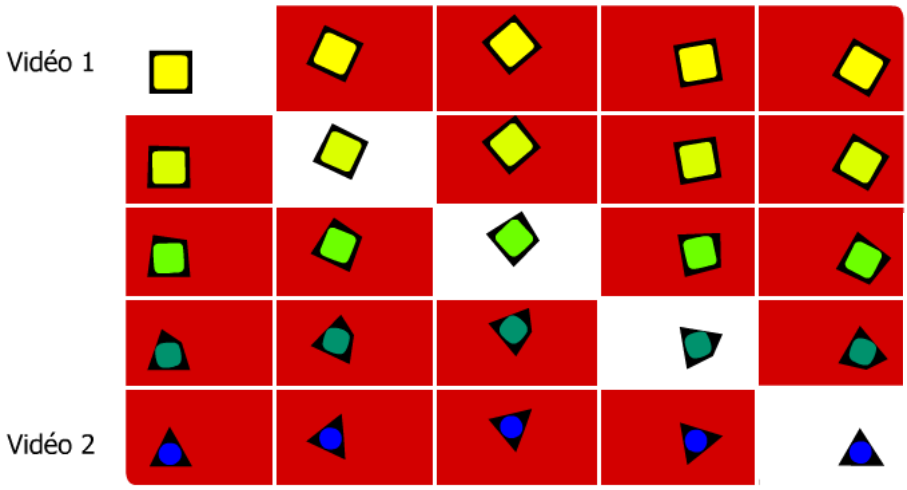

Fig. 6 : Schéma explicatif du morphing entre deux vidéos – un morphing est effectué entre chaque couple d'images correspondantes des deux vidéos et une seule étape est gardée à chaque fois.

Il est impossible de demander à l'utilisateur de faire les correspondances entre toutes les images des deux vidéos, car avec 25 images par seconde, cela représente 1500 images pour une minute et il serait donc trop long et fastidieux à réaliser même pour un morphing de seulement une minute. La solution la plus sûre est de demander à l'utilisateur de faire les correspondances seulement entre certaines images et que le logiciel fasse l'interpolation entre ces points. L'inconvénient de cette méthode est qu'elle reste très longue à réaliser et si l'on ne prend pas beaucoup d'étapes elle ne sera pas très précise. C'est cette méthode qui est implémenté dans tous les logiciels de morphing actuellement car c'est la méthode la plus robuste.

Nous proposons une nouvelle méthode beaucoup plus rapide pour l'utilisateur. Ce dernier clique des points de correspondance uniquement entre la première image de chacune des deux vidéos, ensuite l'application réalisera un tracking de ces points et le morphing pourra se faire directement.

Algorithmique

Au lieu de parcourir chaque pixel de l'image résultat et de chercher dans quel triangle il se trouve, on parcourt chaque triangle de l'image résultat et on la dessine au fur et à mesure. Actuellement, nous parcourons tous les points de la boite englobante de chaque triangle et l'on teste si l'on est à l'intérieur du triangle. Une petite amélioration serait d'utiliser les algorithmes de synthèse d'image pour ne parcourir que les pixels à l'intérieur du triangle.

Pour calculer les pixels correspondant au pixel du triangle de l'image résultat dans les deux images sources, on peut utiliser plusieurs méthodes. On peut se baser soit sur un espace vectoriel, soit sur des méthodes calculant un coefficient par rapport à chaque sommet du triangle.

Nous avons fini par opter pour une méthode basée sur des coefficients pour sa résistance aux cas particuliers. Nous avons calculé ces coefficients à partir de l'équation du plan.

$$\alpha(x,y) = ax + by + c$$

A, B, C sont les points du triangle.

<u>cas 1</u> : si le point P est sur le sommet A, alors

$$\alpha(x_A, y_A) = 1 \qquad a * x_A + b * y_A + c = 1$$

$$\alpha(x_B, y_B) = 0 \qquad a * x_B + b * y_B + c = 1$$

$$\alpha(x_C, y_C) = 0 \qquad a * x_C + b * y_C + c = 1$$

On obtient :

$$b = \frac{x_{CB}}{(x_{CB} y_{BA} - y_{CB} x_{BA})}$$

$$a = -b\left(\frac{y_{CB}}{x_{CB}}\right)$$

$$c = -ax_B - by_B$$

Un cas particulier se produit avec les droites verticales et horizontales (une variable devient nulle et l'on ne peut plus résoudre l'équation). Nous devons donc résoudre 3 fois le système en se

basant sur les points dans un ordre différent (les 3 cotés du triangle ne pouvant être uniquement verticaux ou horizontaux!).

cas 2 : On fait de même en supposant le point P sur le sommet B

$$b = \frac{x_{CB}}{(x_{BA}y_{AC} - y_{BA}x_{AC})}$$

$$a = \frac{(1 - by_{BA})}{x_{BA}}$$

$$c = 1 - ax_A - by_A$$

cas 3 : Puis enfin sur le sommet C

$$b = \frac{x_{CB}}{(x_{AC}y_{CB} - y_{AC}x_{CB})}$$

$$a = \frac{(-1 - by_{AC})}{x_{AC}}$$

$$c = -ax_C - by_C$$

On réalise une triangulation sur l'image de départ et une triangulation sur l'image d'arrivée; car comme les points vont se déplacer, la triangulation ne sera pas forcement la même sur l'image de départ et sur l'image d'arrivée. De plus cela peut également limiter l'apparition des segments des triangles vu que ceux-ci ne seront de cette façon pas forcement aux mêmes endroits sur les deux images. De plus, vu le nombre de points que l'on utilise pour un morphing, la triangulation n'est pas trop coûteuse.

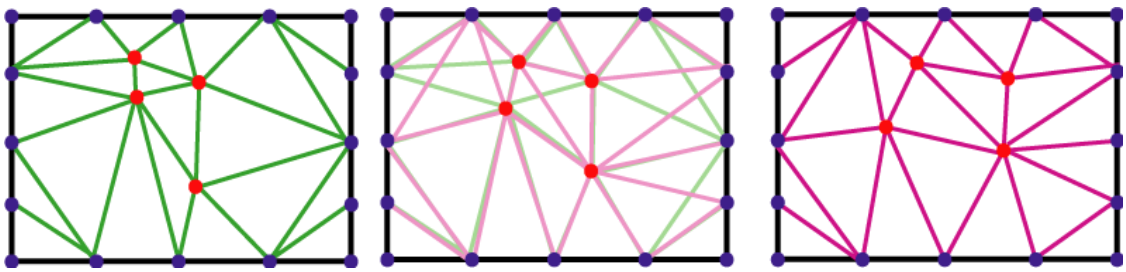

Fig. 7 : Une triangulation différente sur l'image de départ et sur l'image d'arrivée

Lorsque l'on va parcourir l'image résultat, on devra donc parcourir tous les triangles de la première image, puis tous les triangles de la deuxième image.

Amélioration

- Il serait intéressant que l'interpolation géométrique des triangles ne soit pas linéaire. Car si un triangle se modifie beaucoup, il serait intéressant pour celui qui le touche, même si ses trois sommets ne changent pas, d'être un peu modifié. L'idée n'est pas de faire comme dans la méthode qui utilise tous les points (et donc très coûteuse en calcul), mais de prendre en compte les trois triangles voisins à chaque triangle. Cela permettrait d'éviter de faire apparaître les segments des triangles lorsqu'il y a de grandes déformations.

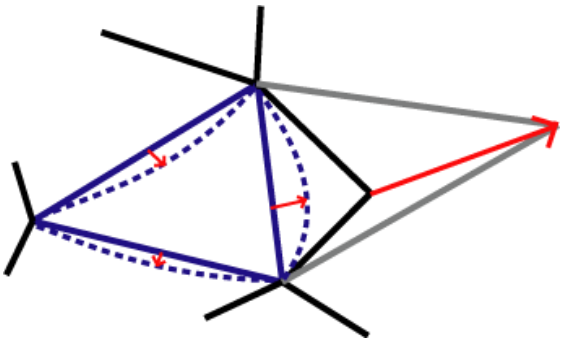

Fig. 8 : Prise en compte des triangles voisins lors du morphing

- Il faudrait faire une triangulation de Delaunay particulière, pour éviter que les points des bords forment des triangles ensemble. Un moyen serait d'utiliser des poids sur les segments lors de la triangulation, mais la librairie utilisée ne propose pas cela. Il faudrait donc l'implémenter, or la triangulation de Delaunay est un algorithme assez complexe et l'amélioration apportée n'est pas si importante.

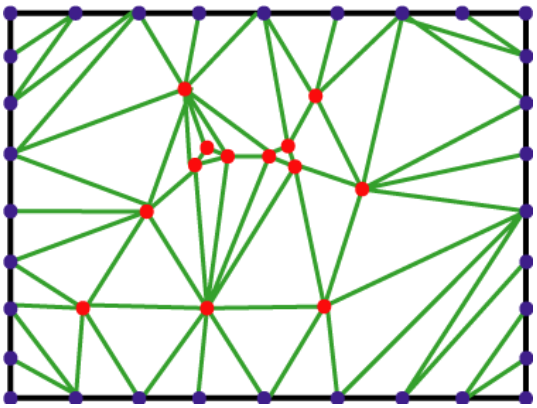

Fig. 9a : Triangulation classique - les points des bords se relient entre eux et resteront donc fixe tout le long, même si un point pas très éloigné se déplace beaucoup

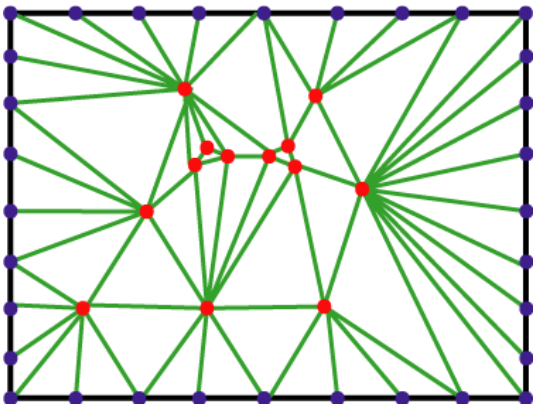

Fig. 9b : Points correctement reliés pour le morphing mais ne respectant pas les règles d'un Delaunay classique

Implémentation

Le code est séparé entre une classe ne contenant que des fonctions statiques, qui possède des fonctions génériques, et une classe qui contient les paramètres du morphing (type d'interpolation, type de tracking), les différentes listes de points (points sur lesquels il faut faire le tracking, points des bords, points fixes) et les liens vers les vidéos.

D) LOGICIELS EXISTANTS

- **XMRM** (Multi Resolution Morphing pour X)

  Logiciel de morphing Open Source sous linux. Logiciel assez complet pour faire un morphing entre deux images.
  http://www.linuxfocus.org/Francais/September2001/article139.shtml

- **XMorph et GtkMorph**

  Projet de morphing open source sous linux et Windows basé sur la librairie Gtk.
  http://xmorph.sourceforge.net/

- **Adobe After Effects**

  Logiciel payant d'édition vidéo pour Windows
  http://www.adobe.com/products/aftereffects/

  Tutoriel pour faire un morphing vidéo sous After Effects
  http://forums.creativecow.net/cgi-bin/new_page_wrapper.cgi?forumid=1&page=http://www.creativecow.net/articles/zwar_chris/morph/index.html

- **WinMorph**

  Freeware pour Windows travaillant entre deux images. Il peut également s'exécuter sous forme de plugin pour Adobe Première, Vegas Video ou Wax, auquel cas il peut travailler entre deux vidéos.
  http://www.debugmode.com/winmorph/

- **Morpheus**

  Freeware en anglais pour Windows
  http://www.morpheussoftware.net/

- **FunMorph**

  Shareware pour Windows
  http://www.funmorph.com/

- **Sqirlz Morph**

  Freeware pour Windows
  http://www.xiberpix.com/

- **Morph X**

  Freeware pour MacOS, permettant de faire un morphing entre deux images et exportation en images ou en vidéo
  http://www.norrkross.com/software/morphx/morphx.php

- **MorphAge**

  Logiciel payant pour MacOS, permettant de faire un morphing entre deux images ou deux vidéos
  http://www.ex-cinder.com/morphage/morphage.html

DETECTION DES POINTS DE HARRIS

A) PRINCIPE

Toujours dans l'optique de minimiser le travail de l'utilisateur, nous proposons une option pour détecter à sa place les points représentatifs de l'une des deux images. Il n'a donc plus qu'à placer la correspondance sur la seconde image.

En effet même si l'utilisateur ne passe pas par cette option il ne peut pas placer les points n'importe où sur l'image. Cependant intuitivement, un être humain va positionner les points à des endroits représentant le contour d'un objet. Or les contours des objets représentent des changements de couleurs et sont donc des endroits clefs (pour le morphing, mais surtout pour le tracking). Ce sont ces points que détecte l'algorithme de Harris.

Il existe de nombreuses méthodes pour détecter les points clefs d'une image, mais l'algorithme de Harris est une valeur sûre, qui fait partie des algorithmes qui donnent les meilleurs résultats dans tous les types d'images.

L'extraction des points de Harris est basée sur la luminosité. Nous avons donc implémenté cette technique classique mais également une extension qui prend en compte la couleur (RGB) lors du calcul du coefficient des points.

Une amélioration serait le filtre de Harris-Laplace, qui pourrait très facilement être rajouté sachant que le filtre de Laplace est une simple convolution (déjà implémenté).

B) ALGORITHMIQUE

La détection des points de Harris est basée sur trois convolutions de l'image :

- une convolution 1D selon X
- une convolution 1D selon Y
- une convolution 2D selon XY

$$M = \begin{bmatrix} Cx^2 & Cxy \\ Cxy & Cy^2 \end{bmatrix} \text{ avec } Cxy = Cx * Cy$$

On doit à partir de là, calculer pour chaque pixel de l'image (sauf les pixels des bords) :

$$Score = det(\mathrm{M}) - k * trace^2(\mathrm{M})$$

avec ***k*** une constante $k = 0{,}04$ (ou $0{,}04 \leq k \leq 0{,}15$)

Le déterminant d'une matrice carrée 2x2 se calcule de la façon suivante :

$$\mathrm{A} = \begin{bmatrix} A & B \\ C & D \end{bmatrix},\ det(\mathrm{A}) = A * D - B * C$$

La trace d'une matrice carrée est la somme de ses éléments diagonaux :

$$trace(\mathrm{A}) = A + D$$

On peut donc calculer le score de la façon suivante :

$$Score = (Cx^2 * Cy^2 - Cxy^2) - k * (Cx^2 + Cy^2)^2$$

L'amélioration pour prendre en compte la couleur part de la matrice suivante :

$$\mathrm{M} = \begin{bmatrix} g(Rx^2 + Gx^2 + Bx^2) & g(RxRy + GxGy + BxBy) \\ g(RxRy + GxGy + BxBy) & g(Ry^2 + Gy^2 + By^2) \end{bmatrix}$$

avec ***g*** une gaussienne.

On obtient donc un tableau de scores (un score pour chaque pixel).

On extrait les maximums locaux :

A	**B**	**C**
E	**E**	**F**
G	**H**	**I**

On parcourt notre tableau de scores, on ne prend en compte que les pixels voisins, dans le schéma ci-joint le point courant est E. Le point E est un maximum local si et seulement si :

$A < E$	et	$B < E$	et	$C < E$
$D < E$	et	.	.	$F \leq E$
$G \leq E$	et	$H \leq E$	et	$I \leq E$

Ensuite tous les points extraits sont triés (en fonction de leur score) pour que l'utilisateur puisse extraire le nombre de points qu'il souhaite parmi ceux détectés.

C) IMPLEMENTATION

Pour la version standard en niveau de gris :

- On parcourt l'image, on calcule les 3 convolutions et l'on peut ainsi directement obtenir le tableau final des scores
- On extrait les maximum locaux
- On les trie

Pour la version couleur:

- On crée 3 tableaux pour les 3 convolutions
- On les filtre avec un filtre Gaussien
- On calcul le tableau final contenant les scores
- On extrait les maximum locaux
- On les trie

L'utilisateur peut maintenant prendre le nombre de points qu'il souhaite dans une liste triée.

Cette liste est sauvegardée dans la classe Harris.

LE TRACKING

Le suivi de points ou suivi de mouvement consiste à établir des correspondances entre deux images. Cette technique est largement employée dans le domaine de l'encodage vidéo pour éviter de sauvegarder toutes les images mais seulement des images clefs, laissant à l'algorithme d'encodage le soin de calculer et reconstruire les images intermédiaires. Dans notre projet on utilise le tracking pour suivre les points cliqués initialement par l'utilisateur et faciliter son travail en automatisant l'application du morphing.

A) LES DIFFERENTES APPROCHES DANS LE SUIVI DE MOUVEMENT

Les algorithmes de suivi de mouvement se divisent en trois catégories principales. On distingue d'abord les algorithmes **basés sur les traits et formes caractéristiques** de l'image. Cela implique l'extraction de ces traits moyennant un filtre pour ensuite établir la correspondance basée sur une particularité géométrique locale.

La deuxième catégorie d'algorithmes est basée sur une **similarité structurelle**. Plutôt que la distribution de points dans les deux images, ces algorithmes considèrent des graphes d'adjacence des traits. Cette méthode part du principe que la topologie d'une image est invariante lors des changements de perspective. Pour cette raison ce type d'algorithme est très puissant et peut être utilisé pour la reconnaissance d'objets.

Néanmoins il nécessite une très grande puissance de calculs, dont le coût n'est pas justifié dans le cas de notre projet. Qui plus est, nous n'avons pas besoin de reconnaître les objets suivis, ni même d'avoir la certitude que les points cliqués correspondent à des contours. Il relève de la décision de l'utilisateur des points sur lesquels il veut baser le morphing. Ainsi nous avons choisi la troisième catégorie d'algorithmes, basée sur une **comparaison colorimétrique**. Ce type d'algorithmes est très sensible aux occlusions mais offre pourtant une précision optimale pour un coût de calculs modéré.

La technique consiste à rechercher une correspondance entre des blocs de pixels des deux images en comparant directement leurs niveaux de gris ou des fonctions qui en dérivent. On distingue donc deux étapes dans la recherche : le parcours des images pour choisir les blocs à comparer et ensuite l'évaluation même de chaque candidat par rapport au bloc d'origine. Il en découle ainsi une combinaison de deux sortes d'algorithmes différents : un algorithme que nous appelons ici « de parcours » et un algorithme (ou fonction) de comparaison.

B) LES ALGORITHMES DE PARCOURS

Il existe une multitude de recherches dans le domaine de la mise en correspondance d'images. L'algorithme le plus simple, le plus fiable et le plus gourmand en ressources est le **Full Search** (recherche exhaustive). Comme son nom l'indique, il procède par comparaison de tous les

blocs dans une zone de recherche définie, pour en choisir celui qui ressemble le plus au bloc d'origine.

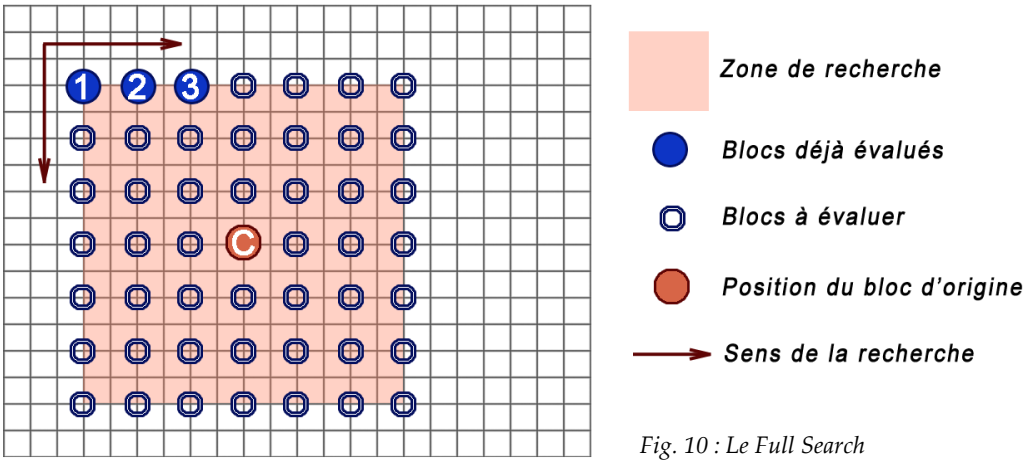

Fig. 10 : Le Full Search

Il est très lent mais donne de bons résultats et était donc le premier à être utilisé. Avec le développement de la vidéo numérique et le besoin croissant de coder et transmettre des flux, il a fallu trouver des algorithmes efficaces et optimisés pour le remplacer. Son défaut principal, qui est en même temps la raison sa précision, est qu'il parcourt tous les blocs possibles, sans tenir compte du fait que si le meilleur bloc est au début, la recherche ultérieure est inutile. Ainsi, tous les autres algorithmes de parcours diffèrent d'abord et principalement par le tri sélectif des blocs à comparer. De ce fait ils sont moins performants en termes de qualité du résultat obtenu, mais en revanche ils sont souvent nettement plus rapides. Pour cette raison ils sont connus sous le nom général d'**algorithmes sous-optimaux** (*sub-optimal*).

Le choix des blocs à comparer repose sur plusieurs critères et postulats différents qui forment quelques groupes d'algorithmes sous-optimaux.

- Certains procèdent par étapes (***signature-based algorithms***). A la première étape tous les blocs sont examinés selon une formule de comparaison peu coûteuse en ressources. A la deuxième étape seuls les blocs les plus prometteurs sont comparés en utilisant une formule plus pointue et ainsi de suite jusqu'à obtention du meilleur bloc.

- D'autres algorithmes de parcours optimisés effectuent une **quantification grossière des vecteurs** de mouvement basée sur la connaissance des limites de l'observateur humain. L'œil humain ne peut percevoir le mouvement au-delà d'une certaine vitesse, ainsi on peut se permettre une baisse de la qualité de l'image dans la zone de l'objet en mouvement. Cela permet de privilégier le centre de la zone de recherche qui correspond au centre du bloc d'origine. Tous les candidats autour de ce centre sont évalués, et seulement quelques uns parmi les blocs plus éloignés. De cette manière on s'assure d'obtenir une bonne correspondance si le bloc recherché est au centre et donc son mouvement est lent et régulier, et un résultat moins précis s'il bouge vite et se trouve plus loin.

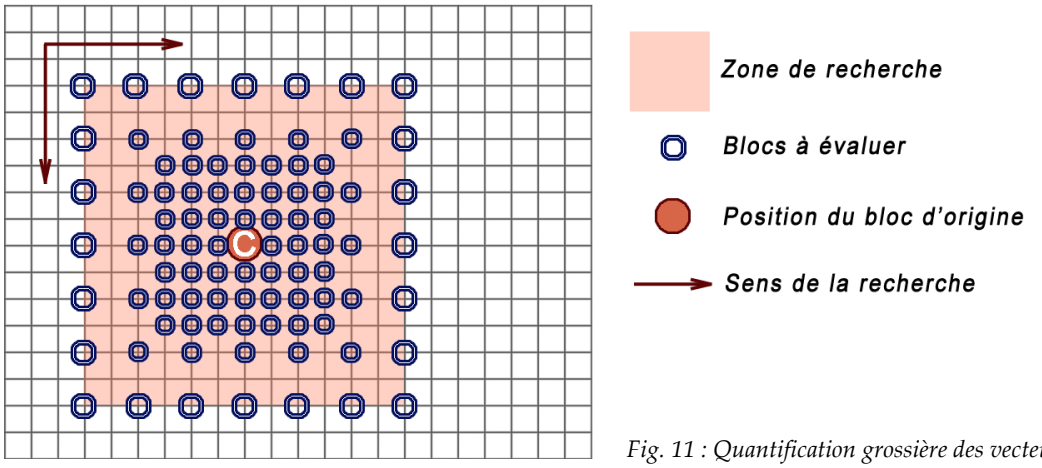

Fig. 11 : Quantification grossière des vecteurs

- Une approche similaire est adoptée par les algorithmes basés sur le **principe de localité**. Ils procèdent également par concentration de la recherche sur une zone choisie, cette fois-ci non pas à la base d'une prédiction du mouvement mais d'après le postulat que les meilleurs résultats se trouvent à proximité des résultats relativement bons. Le problème de ces algorithmes réside dans le risque de converger vers un minimum local plutôt qu'un minimum global. On peut palier à ce problème en considérant plusieurs minimums locaux durant les premiers stades de la recherche, pour en choisir le meilleur à la fin.

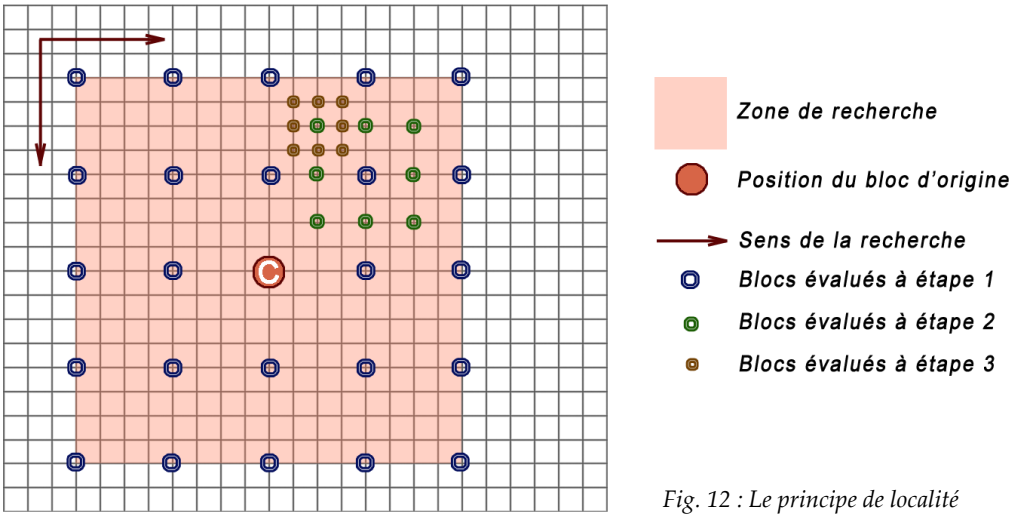

Fig. 12 : Le principe de localité

- Ensuite nous pouvons citer ici les algorithmes qui reposent sur **l'homogénéité spatiale et/ou temporelle** des images. Ainsi le vecteur de mouvement d'un bloc peut être prédit en fonction du déplacement des blocs voisins. Il peut également être prédit sur la base de son déplacement antérieur.

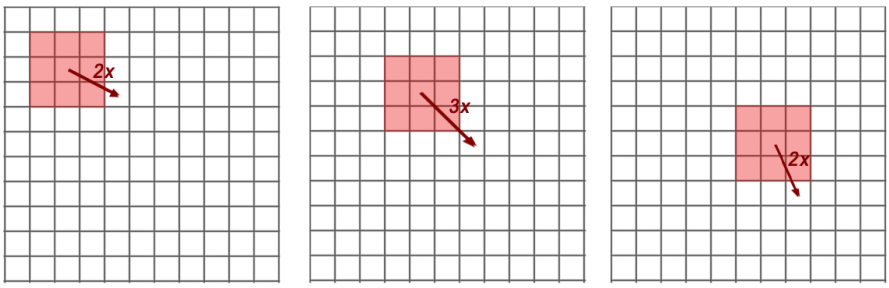

Fig.13 : Prédiction du mouvement d'un bloc en fonction de son déplacement

- Le dernier groupe d'algorithmes que nous avons étudiés sont les **algorithmes pyramidaux** de vecteurs ou de niveaux de gris. Ce sont des algorithmes à plusieurs étapes qui considèrent au départ l'image décomposée en blocs grossiers. Le meilleur d'entre eux est choisi (selon son vecteur de déplacement ou son niveau de gris) et la résolution est augmentée d'un cran, jusqu'à ce que l'on revienne à l'échelle 1 des pixels.

C) ETUDE COMPARATIVE DES ALGORITHMES DE PARCOURS

L'implémentation et le test de tous ces algorithmes n'étant pas à notre portée, ni le but de notre projet, nous nous sommes fiés aux études comparatives effectuées par les chercheurs dans ce domaine pour choisir et tester les algorithmes qu'on considère les plus adaptés à notre cas.

Nous avons éliminé d'emblée les algorithmes qui utilisent l'homogénéité de l'image car nous ne disposons pas des informations des blocs voisins.

Par ailleurs les autres algorithmes sont extrêmement difficiles à comparer, étant donné que les résultats de chacun diffèrent selon les séquences de test et qu'il est souvent difficile d'établir le bon rapport qualité/complexité.

La plupart des algorithmes procèdent par déplacement de la recherche vers la zone de meilleur score en diminuant progressivement le pas. Tels sont le Three Step Search (TSS) et le 2D Logarithmic Search (2DL). Ces deux algorithmes prennent un nombre de points spécifique à chacun, dans une disposition prédéfinie (9 points en grille pour le TSS et 5 points en « + » pour le 2DL) et à chaque étape déplacent le centre de la nouvelle recherche sur le point de meilleur score de l'étape précédente. Le TSS divise le pas par 2 à chaque étape, tandis que le 2DL ne diminue la distance que si le meilleur point est déjà au centre. Le TSS s'arrête quand la distance devient plus petite ou égale à 1, tandis que le 2DL évalue encore une fois les blocs tout autour.

Très semblable à ces deux algorithmes, le Cross Search (CS) ne diffère que par la disposition des points au début, en « x » au lieu de « + ». A la fin, comme le 2DL il évalue tous les blocs autour, en forme de « + » cette fois.

Un quatrième algorithme, le Spiral Search (SS) est basé sur le regroupement du TSS et du 2DL. Il prend neuf points, le centre puis quatre points en « x » à distance D autour et 4 autres en « + » à distance D/2. Ensuite il procède pareil, en déplaçant le centre de la recherche et diminuant la distance à moitié.

Le Four Step Search exploite également le principe de localité, avec cette différence que lors des déplacements du centre de recherche il n'évalue pas les points tout autour, mais seulement ceux qui se trouvent dans la même direction. C'est-à-dire que s'il vient d'effectuer un mouvement en haut et à gauche, il utilisera pour la comparaison les blocs en haut et à gauche du nouveau centre.

Ce principe de privilégier le mouvement est repris par d'autres algorithmes de parcours, comme le Orthogonal Search (OS) ou le « One at A Time » Search (OTS). Ils effectuent encore moins de comparaisons, au détriment de la qualité de leurs résultats. Ils privilégient complètement la direction de la meilleure correspondance trouvée et ne recherchent pas tous les blocs autour. Le OS teste d'abord 3 points à l'horizontale, puis place le centre de recherche sur le meilleur et teste 3 points à la verticale. Les mêmes évaluations sont répétées en diminuant la distance jusqu'à ce qu'elle devienne plus petite ou égale à 1. Le OTA ressemble au OS par le fait d'avoir une étape horizontale et une verticale. En revanche il se déplace sur la même direction (horizontale d'abord, puis verticale) tant que le point suivant dans la même direction a un score plus petit que le précédent. Ensuite il change de direction, il répète la même logique et s'arrête dès qu'il trouve le point de moindre score (c'est-à-dire celui qui a la moindre différence colorimétrique).

Un algorithme qui diffère un peu plus de ceux déjà cités est le Binary Search (BS), utilisé notamment pour l'encodage du format MPEG4. Il sépare initialement l'image en 9 zones qui incluent le centre, les coins de la zone de recherche et les milieux de chaque bord de la zone (voir schéma ci-après). Les blocs centrés dans ces points sont comparés avec le bloc d'origine selon un critère simple et la zone qui contient le meilleur point est parcourue avec un Full Search. Le seul souci de cet algorithme est le fait que les pixels entre les 9 zones ne sont jamais considérés, mais autrement il propose un bon rapport qualité/temps CPU.

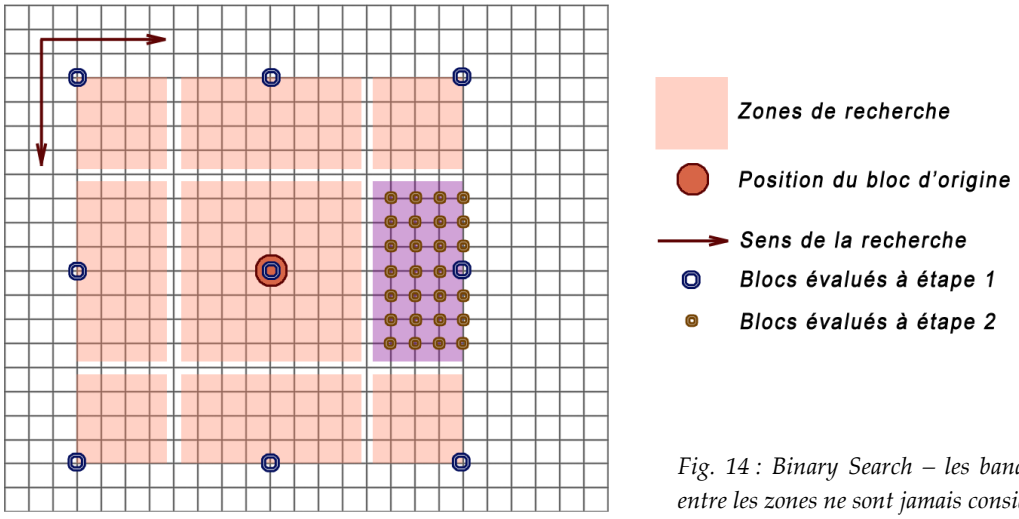

Fig. 14 : Binary Search – les bandes blanches entre les zones ne sont jamais considérées

Il existe un autre groupe d'algorithmes, appelés Greedy Algorithms, ce qui veut dire littéralement « gourmands », parce qu'ils considèrent chaque meilleur résultat comme un indice pour se déplacer immédiatement dans cette direction. En général ils testent successivement toutes les directions à partir du centre dans un ordre prédéfini et changent de centre de recherche dès que le point évalué est meilleur. Les multiples Greedy Algorithms (nommés de A à F) diffèrent principalement par le choix de l'ordre des directions et le coefficient de diminution du pas. Parmi

eux celui qui est réputé pour donner les meilleurs résultats est le F, qui évalue en priorité les directions horizontales, basé sur le postulat que dans une séquence vidéo le mouvement horizontal est plus souvent rencontré. A chaque étape la distance de recherche est diminuée de 25%, ce qui fait qu'il converge moins vite et a donc moins de chances de tomber sur un minimum local.

Finalement, plus complexes mais apparemment efficaces, sont les algorithmes de recherche pyramidaux ou hiérarchiques. Lors de la première étape ils subdivisent la zone de recherche en quelques zones et font une moyenne de leur couleur pour la première évaluation ou bien ne prennent qu'un pixel sur deux ou sur trois. A chaque étape ils augmentent la précision pour les zones de meilleur score de l'étape précédente.

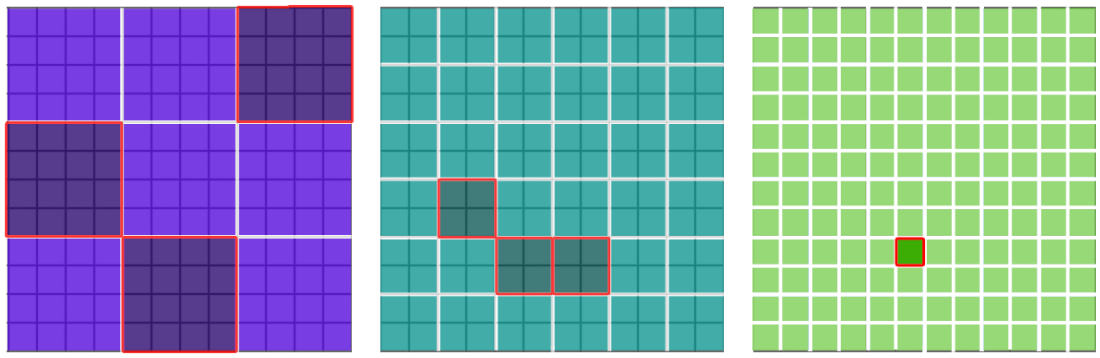

Fig. 15 : Algorithme de recherche hiérarchique

Parmi tous ces algorithmes, suivant les différentes séquences et conditions de test (voir bibliographie) le Three Step Search donne souvent le meilleur résultat après le Full Search, tout en étant plus rapide que les algorithmes hiérarchiques. Le Four Step Search est plus rapide que lui mais moins précis. Ensuite, avec une bonne performance au niveau du temps, viennent les Greedy Algorithms, parmi lesquels le F est le meilleur. Le One at A Time Search est souvent en queue des classements au niveau de la qualité et en tête par rapidité. Pour des séquences rapides le 2DL a donné de bons résultats, suivi immédiatement du Orthogonal Search. Pour ce qui est des algorithmes hiérarchiques, généralement plus lents que les autres, ils atteignent de meilleurs résultats, souvent proches du Full Search[4].

D) IMPLEMENTATIONS ET TESTS DES ALGORITHMES DE PARCOURS

Sur la base de ces résultats nous avons choisi deux algorithmes différents pour les implémenter et tester dans notre application.

- Le premier est le **Spiral Search**. Ce choix est motivé autant par sa performance reportée dans les études, que par le fait qu'il combine quelques approches communes

4. *Pour les sources des tests voir Bibliographie::Tracking, ainsi que les résultats comparatifs en annexe*

à plusieurs autres algorithmes comme le Three Step Search ou le 2D Logarithmic Search qui montrent également un bon score général.

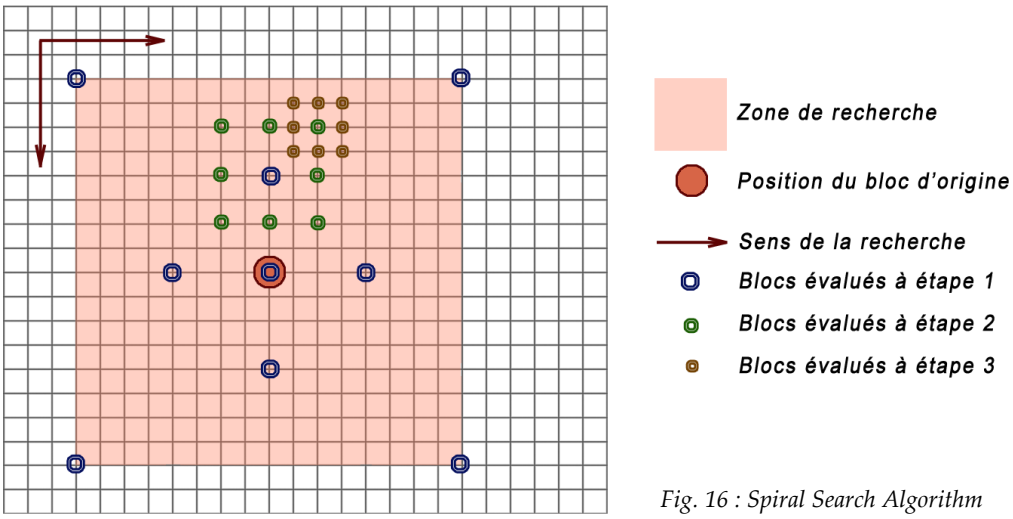

Fig. 16 : Spiral Search Algorithm

Par rapport au Full Search, le Spiral Search s'est avéré extrêmement performant en termes de temps de calcul. Néanmoins, les résultats obtenus ne nous ont pas entièrement satisfaits. Etant donné qu'il converge très rapidement vers un minimum local, particulièrement au début, la qualité du résultat dépend très fortement de la concordance de la direction de mouvement avec celle de convergence de l'algorithme.

- Ce test nous a guidés dans le choix du deuxième algorithme testé – le **Greedy Algorithm F**. Il est reporté meilleur dans son groupe d'algorithmes qui privilégient une direction de recherche. Dans le cas du Greedy F, la direction privilégiée est l'horizontale de droite à gauche, car il a été postulé que c'est une direction plus probable pour le mouvement dans une séquence vidéo. Parallèlement à cela l'algorithme exploite jusqu'au bout la direction verticale dans les deux sens. Malgré la logique moins aléatoire de cet algorithme par rapport au Spiral Search, il s'est avéré aussi exigeant des conditions particulières, notamment un sens prédéfini du mouvement. Il a l'avantage de diminuer le rayon de recherche progressivement au lieu de le diviser par deux à chaque étape, mais ne garantit toujours pas un succès à 100%.

Dans le cadre d'utilisation du tracking dans notre application, il est indispensable de garantir un succès optimal, car la perte d'un point aura une répercussion sur le reste du morphing. Ainsi, après cette étude, il était évident pour nous à cette étape que le Full Search était seul à nous donner des résultats convenables. Ainsi nous avons orienté nos efforts d'optimisation plutôt vers la deuxième partie de la recherche – la comparaison des blocs.

E) ALGORITHMES DE COMPARAISON

La comparaison des blocs s'effectue sur la base d'un critère de correspondance (fonction de distorsion). Le nombre d'algorithmes utilisés est moins important que celui des algorithmes de parcours, mais il existe de nombreuses variations et optimisations de chacun d'entre eux.

Pour nos recherches et tests nous avons implémenté les trois principales fonctions de distorsion – la moyenne des distances absolues (Mean Absolute Differences), la moyenne des distances au carré (Mean Squared Differences) et la classification des pixels par différence (Pel Difference Classification). Nous avons également testé une modification de la MAD – la moyenne absolue des différences des lignes, ce qui était une étude préliminaire à l'implémentation de la dernière formule – celle des projections intégrales (Integral Projections).

- La première formule testée, **la moyenne des distances absolues**, est la plus simple connue. Pour chaque bloc candidat on effectue une comparaison des niveaux de gris avec le bloc d'origine. On somme les différences, absolues pour que les valeurs ne s'annulent pas, et l'on normalise le résultat. Le candidat avec le moindre score, c'est-à-dire la moindre différence de niveaux de gris, est choisi.

$$\frac{1}{mn}\sum_{p=1}^{m}\sum_{q=1}^{n}\left|A(p,q)-B(p,q)\right|$$

- **La moyenne des distances au carré** fonctionne d'une manière très similaire, avec la seule particularité que les différences sont mises au carré avant d'être sommées. Ceci est sensé apporter plus de précision aux calculs, mais il ne faut pas oublier que cela oblige également d'ajouter des opérations de multiplication – une par pixel. Par ailleurs, Dianysian et Baker ont trouvé que la différence de qualité des résultats par rapport au MAD était trop infime pour être perçue.

$$\frac{1}{mn}\sum_{p=1}^{m}\sum_{q=1}^{n}\left(A(p,q)-B(p,q)\right)^{2}$$

- La troisième formule implémentée, la **Pixel Difference Classification**, n'utilise pas une moyenne. Elle compare chaque pixel des deux blocs et le déclare « suffisamment similaire » ou non, par rapport à un seuil de similarité prédéfini. Ensuite le bloc qui comporte le plus de pixels correspondants à ceux du bloc d'origine, est choisi comme le meilleur candidat. Le fait de comparer indépendamment tous les pixels fait de cette fonction la plus rigoureuse, mais aussi celle qui a plus de chance de passer à côté du meilleur bloc dans certains cas particuliers comme le passage d'un objet de l'ombre à la lumière, où une moyenne de tous les pixels aurait plus de chances de rattraper la différence.

$$\sum_{q=1}^{n}\sum_{p=1}^{m} ord\left(\left|A(p,q)-B(p,q)\right|\leq s\right)$$

où ***s*** est le seuil et ***ord***(*expression*) retourne 1 si *expression* est vrai et 0 sinon.

Certains chercheurs ont proposé une amélioration de cet algorithme par l'utilisation d'un seuil de similarité variable, qui commence haut et diminue progressivement en éliminant des blocs au fur et à mesure. Nous avons trouvé cette solution intéressante, mais elle travaille toujours dans le sens d'un meilleur résultat plutôt que d'une recherche plus rapide.

- Nous avons enfin étudié une fonction de distorsion quelque peu singulière – celle des **projections intégrales**. Elle nous a offert la solution que nous cherchions, notamment une optimisation des calculs au niveau de la comparaison des blocs. Cette méthode consiste non pas à comparer pixel par pixel les deux morceaux d'image, mais ligne par ligne et colonne par colonne. Il est bien entendu que ce gain de calculs est au détriment de la précision de la comparaison, effectuée ici entre deux groupes de pixels et non sur un couple de pixels.

$$\sum_{p=1}^{m}\left|\sum_{q=1}^{n}A(p,q)-\sum_{q=1}^{n}B(p,q)\right| + \sum_{q=1}^{n}\left|\sum_{p=1}^{m}A(p,q)-\sum_{p=1}^{m}B(p,q)\right|$$

F) RESULTATS DES TESTS DES ALGORITHMES DE COMPARAISON

Les résultats de toutes ces fonctions ont été pondérés par un coefficient qui ne figure pas dans les formules ci-dessus et qui est la distance du bloc candidat par rapport au bloc d'origine. Nous avons considéré que plus le bloc comparé est proche du bloc de départ, meilleur est son score. Il a fallu gérer le poids de la distance par rapport au score de la comparaison des niveaux de gris. Nous l'avons finalement ramené vers un chiffre généralement compris entre 0 et 1. Ce choix est basé exclusivement sur des expériences et n'est pas également judicieux dans tous les cas de figure. Nous devrions noter ici que nous avons considéré la possibilité de privilégier la nouvelle position prédite pour le meilleur bloc en fonction de son mouvement antérieur, mais pour des raisons d'ergonomie du code nous avons décidé de reporter cet ajustement des calculs dans la partie de gestion d'erreur.

Concernant la performance de chacune de ces quatre fonctions, la logique veut que le MAD soit le plus rapide et le MSD le plus précis. Cependant, nous n'avons pas pu constater une différence marquante ni au niveau du temps CPU, ni au niveau des résultats, à l'exception du PDC

qui nécessitait des réglages différents pour le poids de la distance. Nous avons conclu que les avantages de l'un ou de l'autre algorithme restent imperceptibles à l'échelle de nos tests[5].

Avec une qualité de résultats équivalente, notre choix devait se baser plutôt sur la rapidité des calculs. A priori les quatre fonctions ont montré la même performance, mais logiquement la fonction des projections intégrales se veut plus optimisée que les autres. Par ailleurs il s'est avéré que même sans ajouter la comparaison des colonnes, la formule reste suffisamment précise et nous amène les mêmes résultats que le MSD. Enfin, son avantage principal est la possibilité de réutiliser les calculs des lignes pour les blocs qui se recoupent. Ceci a motivé notre choix, que nous détaillons dans la partie suivante.

G) IMPLEMENTATION FINALE

Comme nous l'avons évoqué, l'algorithme de parcours retenu pour le tracking était le Full Search et nous avons cherché une formule de comparaison optimisée à utiliser avec celui-ci. La formule la plus rapide était celle des projections intégrales, qui était aussi celle susceptible de se prêter à d'avantage d'optimisation, car toutes les autres effectuaient une comparaison pixel par pixel et avaient donc forcément besoin de parcourir tous les pixels des blocs. La fonction des projections intégrales, au contraire, est basée sur une comparaison des lignes, ce qui lui donne son avantage principal – la possibilité de réutiliser partiellement les calculs pour les lignes qui se superposent.

Cet avantage, peu important lors de l'utilisation d'un algorithme de parcours sous-optimal, est d'autant plus intéressant pour nous, étant partis sur une recherche complète. Si chaque bloc suivant a une partie commune avec le précédant, comme lors du Full Search, une partie de la somme de ses lignes serait déjà calculée. Ainsi, pour un bloc de 10x10 pixels, au lieu de faire 10 additions par ligne pour chaque nouveau bloc, on en fait 10 pour le premier et ensuite on peut se contenter d'une soustraction et une addition à chaque étape.

5. Voir la partie 5 - Résultats :: Tracking

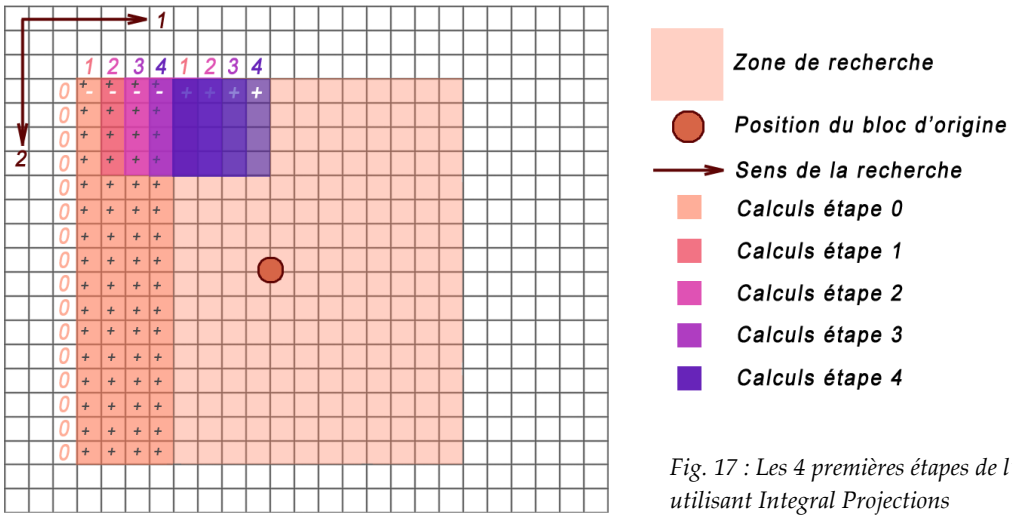

Fig. 17 : Les 4 premières étapes de l'algorithme utilisant Integral Projections

Au final, dans une zone de recherche de 40x40 pixels, avec des blocs de comparaison de 10x10 pixels, on aura fait (40x10 + 40x30x2x10)=24400 opérations, au lieu de 30x30x100 = 90000 opérations, soit 72,8% d'optimisation.

Nous avons donc développé une fonction hybride entre algorithme de parcours et algorithme de comparaison, qui, à la différence des autres algorithmes implémentés, ne peut pas être testée dans des combinaisons différentes, car son optimisation a nécessité une imbrication des deux algorithmes. Par sa rapidité exceptionnelle et son taux de réussite égal aux autres, il est le principal algorithme utilisé et le seul qui permet une exécution du programme en un temps raisonnable. Cependant, pour permettre des tests de comparaison, nous avons laissé à l'utilisateur du plugin la possibilité de choisir sa combinaison d'algorithmes parmi ceux implémentés.

H) Gestion d'erreur

Il faut noter qu'aucun des algorithmes n'a donné un taux de réussite de 100%. Les algorithmes basés uniquement sur les informations des pixels sont particulièrement sensibles aux occlusions, rotations, changements d'échelle, changements de luminosité, ainsi que le bruit et les problèmes de résolution. Certains de ces problèmes peuvent être surmontés par des tests supplémentaires, comme la rotation par exemple. Pour chaque bloc comparé il faudrait effectuer une rotation en mémoire, puis comparer, puis tourner encore et comparer. En fonction du pas de rotation et de l'angle maximal qu'on vérifie, cela ajoute un certain nombre de comparaisons. Le même type de corrections peut être appliqué pour la luminosité et avec moins de succès pour le changement d'échelle.

Mais cette augmentation du nombre de calculs a besoin d'être justifiée. Or dans le cas de notre projet le tracking n'est même pas l'opération principale, mais un moyen au service du morphing, qui lui aussi prend du temps à chaque étape. Il est donc peu recommandé d'alourdir la charge. L'autre solution possible serait de compter sur l'utilisateur pour la vérification de tous les points, mais cela irait à l'encontre du concept du morphing automatisé. Par conséquent nous avons décidé de mettre en place un système semi-automatique de gestion d'erreur, où la détection des points perdus se fait automatiquement et leur correction est laissée aux soins de l'utilisateur.

La première idée pour détecter les erreurs était de se baser sur le score de comparaison de chaque bloc. Tant qu'il est inférieur à un seuil défini, le bloc serait considéré bon et une fois le seuil dépassé on signalerait une erreur. Une des difficultés de cette méthode était de fixer le seuil. Les tests nous ont montré que le score pouvait largement varier selon la séquence utilisée et qu'il était impossible de définir un seuil en avance. En revanche, supposant que la première recherche renvoie toujours un succès, le premier score trouvé pourrait servir de base pour se rendre compte de la grandeur des scores à venir et déterminer un seuil. Mais le problème principal, venant toujours des résultats des essais, était qu'un changement brusque de score ne reflétait pas toujours une perte du point et inversement, un bloc qui ne correspond pas à la recherche peut fournir un score normal à la comparaison. Ceci est valable surtout dans le cas où le décalage s'effectue progressivement et les blocs trouvés s'éloignent petit à petit du bon résultat.

Il nous a donc semblé une meilleure idée, plutôt que de comparer chaque score à une valeur numérique quelque peu aléatoire, d'utiliser pour comparaison la valeur de ce même score quelques images auparavant. Toutes les 10 images nous avons installé une comparaison supplémentaire avec le meilleur bloc de la 10ème image précédente. Ceci permet de dire pour un bloc donné s'il a beaucoup changé en 10 itérations. Nous prenons en compte le fait qu'il change nécessairement et que le score le plus récent ne peut jamais être égal à celui 10 images auparavant. Mais nous considérons également que cet intervalle, qui représente en général une demi-seconde, n'est pas suffisant pour qu'un changement radical s'opère et donc si on observe une différence de score trop importante on peut reconnaître une erreur.

Pour compléter cette méthode qui reste imparfaite, nous avons dirigé notre réflexion dans la direction du déplacement effectué par les points. Nous avons décidé de calculer, au fur et à mesure de l'avancement du point, une moyenne de son déplacement qui sert de référence pour chaque nouvelle position trouvée. Si celle-ci s'avère trop éloignée, elle est considérée fausse. Il faut noter que pour le calcul de la moyenne on pondère spécialement le déplacement des deux dernières étapes pour pouvoir gérer un changement soudain de vitesse de mouvement d'un objet, sa mise en marche ou son accélération. De plus nous relevons l'éventuelle accélération dans les deux derniers mouvements pour prévoir une marge proportionnelle pour le nouveau déplacement.

La gestion d'erreur ainsi mise en place permet d'établir un contrôle équilibré de la part de l'utilisateur. D'une part cela lui permet d'apporter des corrections régulières sur les points perdus. (à cette occasion rien ne l'empêche de corriger également d'autres points). D'autre part, s'il considère que l'intervention n'est pas nécessaire, il peut directement valider et continuer. Au final, grâce au Full Search combiné à la gestion d'erreur, nous avons pu éliminer la possibilité de perdre un point pendant le suivi, mais aussi de minimiser le déplacement de celui-ci autour de sa position précise.

CONCEPTION TECHNIQUE & FONCTIONNALITES

LA VIDEO

A) PRESENTATION

La gestion de la vidéo est assez complexe pour de multiples raisons.

Avant tout, il existe de très nombreuses façons de gérer la vidéo, avec plus ou moins de compression, avec ou sans perte. De plus, ces formats, pour des raisons de compression, ne travaillent pas tous dans le même espace colorimétrique (YUV ou RGB).

Enfin, suivant les besoins, on peut travailler seulement en niveaux de gris, ou bien en couleur ou même éventuellement gérer l'alpha !

Voici donc tous les paramètres qui peuvent varier :

- dimensions
- profondeur des pixels (depth) et bpp (bits per pixel) (8, 16, 24, 32 bits)
- espace colorimétrique (YUV, YUYV, RGB, RGBA, ABGR, etc...)
- nombre de composantes : niveau de gris (1), couleur(3), gestion de l'alpha (+1)

Le fait par exemple de changer d'espace colorimétrique n'est pas très grave puisque cette opération est sans perte, mais il a un coût. La profondeur des pixels est très complexe à gérer, puisse que cela impose de travailler avec des types de variables différents. Enfin en ce qui concerne le nombre de composantes, on ne peut pas toujours appliquer les mêmes effets.

Tous ces paramètres sont donc très complexes à gérer dans leur intégralité.

La deuxième difficulté est que le travail en vidéo est toujours très coûteux en temps de traitement mais aussi en espace disque et nécessite donc d'être optimisé au maximum des deux cotés. Par exemple, on ne peut pas charger toute une vidéo en mémoire et cela oblige d'aller lire en permanence dans un fichier. Or l'accès au disque dur est lent et la compression est presque toujours basée sur des images clefs. Ainsi pour accéder à une image il faut finalement décompresser toute la vidéo entre les deux images clefs.

B) LE CHOIX DE GSTREAMER

La troisième difficulté est la gestion de la multitude de codecs et la complexité de paramétrage de chacun. Il est absolument impossible de gérer tous les codecs soi-même et il faut

donc recourir à une librairie. Sous linux la vidéo et même le multimédia en général sont encore en développement.

Il existe beaucoup de lecteurs vidéo capables de décompresser de la vidéo et de l'afficher avec de multiples optimisations, mais elles sont plus rares à permettre de manipuler réellement nous même le flux vidéo. Nous en avons vu principalement deux: Xine et GStreamer, ainsi que les librairies AVIFile et ffmpeg. Après une grande phase de recherche sur les libraires existantes, les librairies utilisés par les différents logiciels traitant de la vidéo sous linux, nous avons choisi GStreamer qui nous a paru plus orienté édition vidéo. Il possède notamment l'extension GnonLin qui pourrait être intéressante pour un futur développement. Mais ce framework est encore en développement et est assez complexe d'utilisation car il n'a pas pour unique but de décoder et d'encoder de la vidéo mais intègre beaucoup de chose en plus (l'affichage, la sortie audio et de multiple manipulation de la vidéo et même la manipulation d'image). En fait GStreamer est orientée temps réel (tous ses plugins d'effets sont censés s'appliquer en direct, etc...).

Elle permet grâce à un système de plugins de gérer les différentes librairies spécifiques à chaque codec. Mais un même codec peut être lu par différentes librairies et donc différents plugins. Or une gestion faisant abstraction complètement de cela et étant capable de choisir le plugin le mieux adapté pour lire une vidéo, faire un transcodage, appliquer un effet, etc. n'est malheureusement pas encore totalement gérée.

Nous avons réalisé de nombreux petits codes en langage C pour arriver à lire de la vidéo avant de l'intégrer dans l'application. Malheureusement nous nous sommes confrontés à de nombreux problèmes. La documentation propose uniquement des exemples pour l'audio, or la même structure pour la vidéo ne marche pas. Nous avons demandé sur de nombreux forums et finalement sur la mailing liste de GStreamer. Nous avons fini par avoir quelques réponses mais les résultats ne sont toujours pas là. Il est assez difficile de comprendre d'où viennent les erreurs car il n'y a pas de problèmes de compilation, il n'y a aucun warning et pas d'erreur non plus à l'exécution. Pourtant l'on voit bien qu'il y a un problème dans la vidéo car dans notre exemple nous avons voulu faire un simple player audio/vidéo et si l'on retire la partie vidéo, l'audio est lu alors que quand celle-ci est là, il n'y a plus de son. Une erreur doit donc se produire dans la partie vidéo qui empêche même l'audio de se lire.

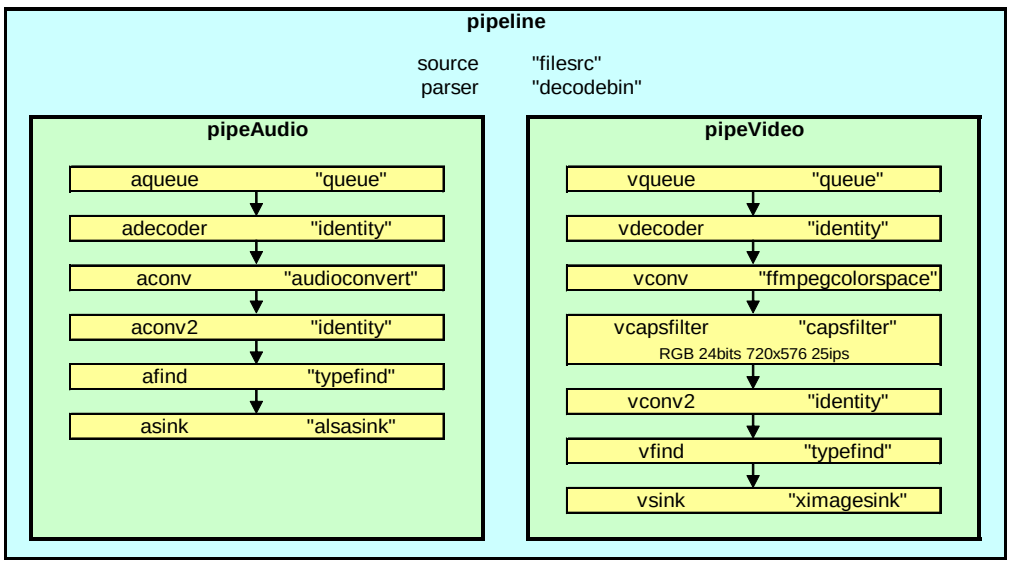

Fig. 18 : Schéma de la structure mise en place dans nos fichiers de tests. Ce schéma possède peut-être des erreurs car la vidéo ne fonctionne pas.

Nous travaillons depuis le début avec une classe qui gère une suite de fichiers images comme une vidéo. La structure des classes réalisée est conçue pour accueillir la lecture vidéo par l'intermédiaire de GStreamer sans changer les autres classes.

C) AUTRES LIBRAIRIES / PLAYERS VIDEO

Étant confronté à de nombreux problèmes avec la librairie GStreamer, nous avons fait quelques tentatives avec d'autres librairies ou players :

Tout d'abord VLC, avec lequel les problèmes ont été d'abord d'arriver à compiler. Ensuite des erreurs se produisaient très rapidement à la lecture. Mais le plus gros problème est d'arriver à récupérer le buffer de l'image. Pour cela il est possible de faire un plugin pour vlc et d'ainsi grâce à ce dernier d'accéder au buffer, mais c'est très complexe à mettre en place.

Nous nous sommes ensuite tournés vers la librairie AVIfile, avec laquelle nous n'arrivons pas à accéder à toutes les images dans le cas d'une vidéo compressée. En réalité nous lisons uniquement les images clés (keyframes). De même, l'encodage se fait toujours en non compressé.

Enfin des exemples d'utilisation de MPlayer avec Qt, nous ont permit de faire un petit player, mais la communication se fait par en mode texte et n'est absolument pas prévu pour cela. Nous nous sommes renseignés mais il ne semble pas possible d'accéder à l'image avec Mplayer, car c'est lui qui gère l'affichage directement.

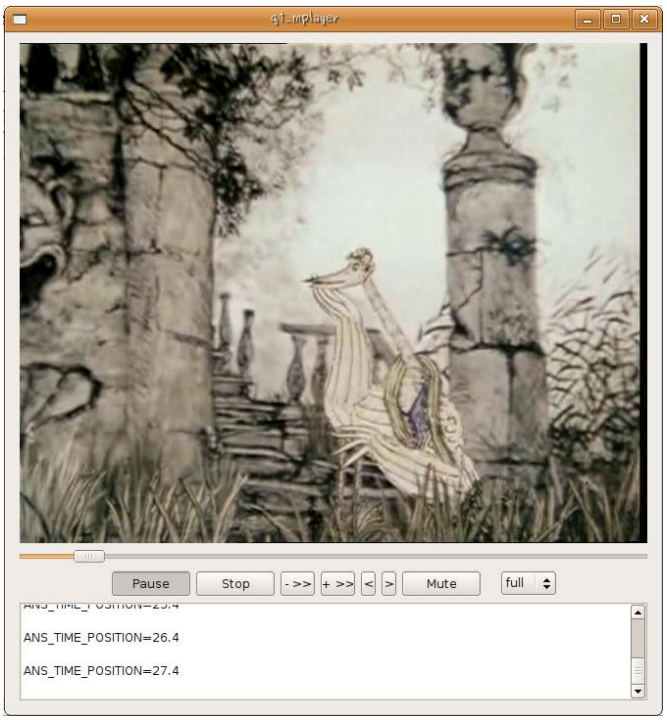

Fig. 19 : Player Qt de test, basé sur MPlayer

GStreamer semble donc bien le plus adapté, même s'il est orienté temps-réel. Les librairies pour lire la vidéo étant spécifiques à des formats (AVIfile, ffmpeg).

PLUGINS

A) PRINCIPE

Pour que l'utilisateur puisse avoir pleinement satisfaction, il faut que l'application propose le plus de fonctionnalités possible. Du coté du montage, la liste des possibilités, bien que très grande, est une liste plus ou moins finie. Mais il n'en est pas de même du coté des effets, où l'on pourra toujours inventer de nouveaux effets. De plus, ceux-ci vont toujours interagir avec l'application de la même façon, c'est pourquoi il est très intéressant de proposer les effets sous forme de plugins. Ainsi, tout utilisateur qui a besoin d'un effet spécifique pourra le développer simplement, sans avoir connaissance de toute l'application. Il pourra ensuite le rajouter dans le dossier « plugins » et celui-ci se rajoutera automatiquement dans l'application principale (sans avoir à recompiler toute l'application). Cela permet donc un développement rapide d'effets qui sont en plus facilement échangeables entre les utilisateurs.

Malheureusement, la librairie utilisée ne fonctionne pas sous Windows et possède quelques particularités sous MacOS. Elle limitera donc le portage de l'application (c'est la seule partie qui limite le portage). Il faudra développer un système similaire pour chaque système d'exploitation.

B) ALGORITHMIQUE

Sous Linux, cela est permis grâce à une librairie en langage C, dans laquelle se trouve une fonction d'entrée qui permet de créer l'objet principal de notre effet. Celui-ci doit hériter d'une classe contenant les fonctions de base que l'on peut ou que l'on doit redéfinir.

Un exemple intéressant est la norme Open FX, qui propose un standard pour que les plugins puissent être compatibles entre différents logiciels. Celui-ci est trop complexe pour être implémenté en si peu de temps, mais montre le fonctionnement complet qu'il faudrait mettre en place.

Notre gestion des plugins est une version très simplifiée qui ne propose pas de gérer tous les cas, mais uniquement les éléments de base.

C) IMPLEMENTATION

Nous avons implémenté une classe de base pour les plugins. Tous les plugins hériteront de cette classe, permettant ainsi de créer pour chaque plugin un objet du même type. Celui-ci contient les fonctions de base en tant que fonctions virtuelles pures et le plugin est obligé de les redéfinir.

Un *#define* est à placer au début du fichier .cpp : il crée une fonction de base en « intern C » qui servira à créer l'objet de base du plugin.

Un plugin vide d'exemple a été réalisé pour servir de modèle à tous les futurs plugins.

- **PluginLoader**: contient les fonctions de base pour charger les plugins.
- **DE_PluginEffect**: classe de base dont héritent les plugins, permettant de récupérer tous les plugins avec un type commun et définissant les fonctions de base.

INTERFACE GENERALE

De l'interface de l'application se distinguent 5 parties principales : la barre de menus, les conteneurs des médias, du moniteur, des effets et de la timeline. Nous allons décrire chacun d'entre eux.

A) MEDIAS

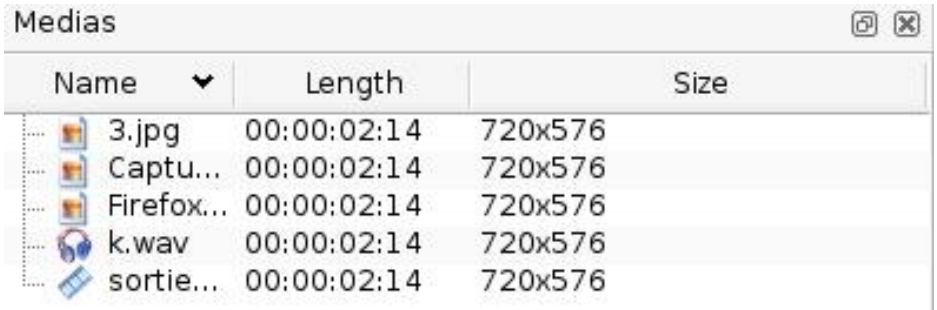

Fig. 20 : Aperçu du panneau des médias

Ce module permet à l'utilisateur de charger un média dans son projet. Un clic droit permet d'obtenir le menu contextuel « Import » qui, une fois cliqué, lui permet de sélectionner un média sur son disque dur. Ce média devra être d'extension standard autorisée par l'application (nous avons vu les formats autorisés précédemment dans ce dossier). Notons l'implémentation des fonctionnalités répondant aux actions suivantes:

- lorsqu'un utilisateur clique (clic droit) dans le module « Media », il peut importer un fichier compatible dans son projet.
- Lorsqu'un utilisateur veut réimporter un média, le répertoire sur lequel il arrive en premier est le répertoire précédemment visité.
- Un utilisateur ne peut pour l'instant importer qu'un seul média en même temps
- Le contrôleur de médias (Media Manager) qui régit les médias dans l'application et que nous verrons plus tard (dans la partie de l'analyse technique) n'accepte qu'une seule entrée d'un média dans la liste. C'est-à-dire que lorsque l'utilisateur voudra importer deux fois le même média, l'application répondra avec un message que le média est déjà présent dans la liste.

Une fois le média importé, il est accessible depuis ce même module « Media ». Des propriétés sont fournies, telles que le nom, la durée et les dimensions. Ainsi l'utilisateur dispose de données utiles sur ce que contient son projet.

Le média peut être déplacé, via une manipulation de type drag 'n' drop vers la timeline. Nous reverrons ce point également dans la partie de développement informatique.

B) Moniteur

Le moniteur fonctionne à la manière d'un écran de visualisation de la séquence. Les boutons de contrôle gèrent ainsi le déroulement temporel du scénario.

La partie suivante expliquera comment nous proposons d'effectuer un rendu grâce à OpenGL, et critiquerons la méthode.

Fig. 21 : Aperçu du panneau du moniteur

C) Panneau d'effets

Les effets sont chargés dynamiquement au chargement du programme. Il y en a trois types pour le moment (les effets sur un média, les effets de transition et les utilitaires). Nous pensons pour plus tard mettre le même mécanisme en œuvre pour la gestion des outils (« trim », « cut » etc.) qui pourront alors êtres personnalisés par les développeurs. L'application ne connaîtra alors plus de limite que celle de leur imagination !

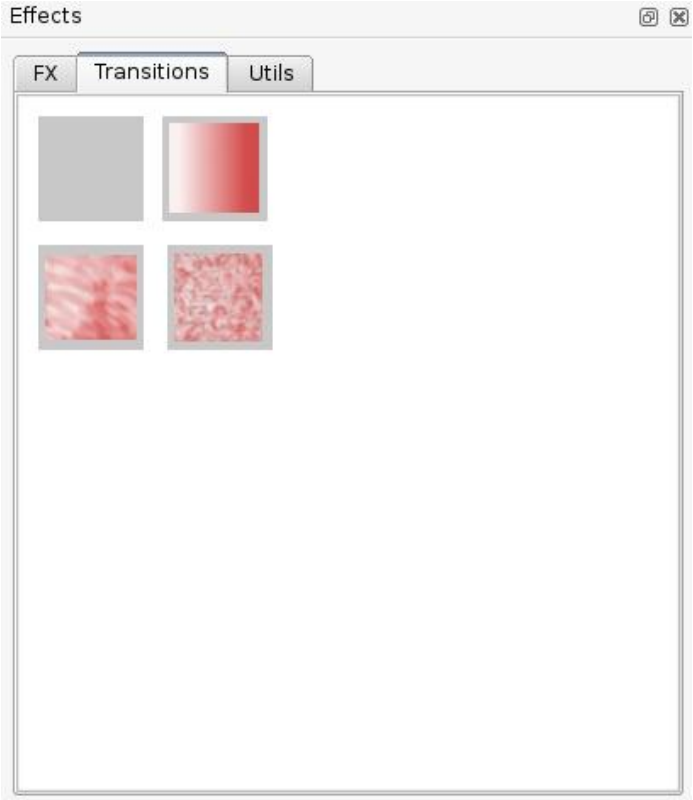

Fig. 22 : Aperçu du panneau du moniteur

D) TIMELINE

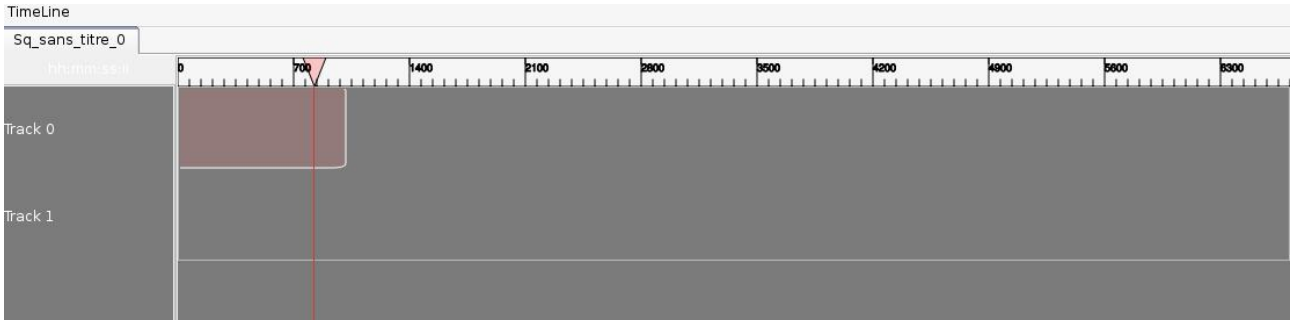

Fig. 23 : Aperçu de la timeline avec un média dessus

La timeline représente le cœur d'une station de montage car c'est sur elle que l'utilisateur va pouvoir manipuler les médias et réaliser son montage, ajouter ses effets, en somme faire jouer sa créativité.

Elle se compose d'une ou plusieurs séquences au gré de l'utilisateur. A chaque séquence on associe un onglet pour pouvoir y accéder rapidement. Chaque séquence possède un nom pour être reconnaissable facilement.

Les séquences sont composées de plusieurs tracks (ou pistes). Chaque track peut être verrouillée pour empêcher que toute modification soit effectuée sur elle, et d'un masque pour pouvoir l'activer (la rendre visible lors de la prévisualisation ou preview du montage) ou non.

Une track est donc l'association d'un espace de montage et d'un espace de contrôle sur le dernier. Une séquence est l'association de ***n*** pistes.

L'utilisateur insère un média sur une track en le glissant/déposant (drag 'n' drop) depuis le module de médias vu précédemment. Il peut juger de la taille occupée par un média en le plaçant puisqu'en le glissant, il peut observer, comme le montre la capture ci-dessous, l'espace qu'il occupe.

La lecture de la timeLine se fait via le panneau de contrôle du module Moniteur, vu précédemment. Cependant, elle dispose d'un outil accessible sur la règle (graduation temporelle faisant état de la durée de la séquence) pour se déplacer dans le montage et rendre une image particulière.

E) MENUS

La barre de menu permet d'accéder à toutes les fonctionnalités proposées par l'application. Nous établissons ci-dessous une liste des menus disponibles ainsi que de leur intérêt.

Menu	Sous-menu	Description	Raccourci	Impl.
File				
	New	Créer un nouveau projet	Ctrl+N	✔

	Open	Ouvre un projet existant	Ctrl+O	✔
	Close	Ferme le projet courant	Ctrl+W	✔
	Save	Sauvegarde le projet courant	Ctrl+S	✔
	Save As	Sauvegarde sous ... (impose le nom de fichier)	Ctrl+Shift+S	✔
	Import	Importe un média dans le panneau des médias.	-	✔
	Export : Current Sequence/ Current Image	Exportation de l'image courante ou de la séquence courante.	-	✔
	Quit	Quitte l'application	Ctrl+Q	✔
Edit				
	Undo	Défait la dernière action	Ctrl+Z	✖
	Redo	Refait la dernière action défaite	Ctrl+Y	✖
	Copy	Copy un média dans le presse papier	Ctrl+C	✖
	Cut	Coupe un média (ajout dans le presse papier et suppression à l'écran)	Ctrl+X	✖
	Paste	Colle un média	Ctrl+V	✖
	Select All	Sélectionne toutes les pistes	Ctrl+A	✖
	Settings	Affiche le panneau de contrôle de l'application	-	✖
Project				
	Settings	Affiche le panneau de contrôle du projet	-	✖
Tools				
	Add Sequence	Ajoute une séquence au projet	-	✔
	Remove current sequence	Supprime la séquence courante du projet	-	✔
	Add track	Ajoute une piste à la séquence courante	-	✔
Effects				
	Special Effects	Liste les effets sur vidéo/image	-	✔
	Transitions	Liste les effets de type transition	-	✔
	Utils	Liste les effets utilitaires	-	✔
Views				
	Medias	Permet d'afficher ou non le panneau de médias	-	✔
	Monitor	Permet d'afficher ou non le moniteur	-	✔

	Effects	Permet d'afficher ou non le panneau des effets	-	✔
	TimeLine	Permet d'afficher ou non la timeline	-	✔
Help				
	Help	Affiche l'aide du logiciel	F1	✔
	About	Affiche les informations sur Studio*DE	-	✔

Cette liste exhaustive nous montre ce qui a été développé ou non au cours du projet. Notamment, nous pouvons indiquer que les méthodes d'annulation d'action n'ont pas été implémentées, ni celles en relation avec le presse papier. Ces fonctionnalités peuvent être utiles et nous comptons les mettre en place rapidement, cependant, elles n'étaient pas prioritaires dans le processus de développement que nous nous somme fixé.

On peut noter également la redondance voulue pour accéder à certaines fonctionnalités. Prenons par exemple l'importation de média. Elle peut s'effectuer par un clic droit dans le panneau média, ou bien à partir du menu File. La création d'un projet peut s'effectuer grâce à ce même menu, ou bien en utilisant des combinaisons de touches clavier (*shortcuts*). Ces éléments ont été mis en œuvre simplement, dans le but de permettre à un utilisateur avancé qui connaît les raccourcis (dont de nombreux sont communs avec d'autres applications) d'aller plus vite dans le processus d'utilisation.

INTERFACE DU PLUGIN MORPHING

L'interface du plugin Morphing offre à l'utilisateur tous les outils pour régler et lancer l'application de l'effet sur le couple de médias choisi dans l'interface principale.

A) FONCTIONNEMENT

Autonome à la compilation, le greffon n'est pas pour autant destiné à exécuter des tâches isolées. Les modifications qu'il effectue doivent avoir une trace et une durée de vie indépendantes de l'interface. Ainsi, au lieu de créer un Morphing à partir de l'interface, qui se retrouverait « orphelin » une fois celle-ci fermée, nous avons décidé de créer l'interface à partir d'un objet Morphing. C'est alors à la charge de l'interface principale de créer l'objet pour initialiser l'interface du plugin. Toutes les modifications sont ensuite appliquées sur cet objet, ce qui fait que si l'utilisateur ferme et rouvre l'interface il retrouvera le morphing (les points, les réglages, etc.) tel qu'il l'a laissé quand il y a travaillé précédemment.

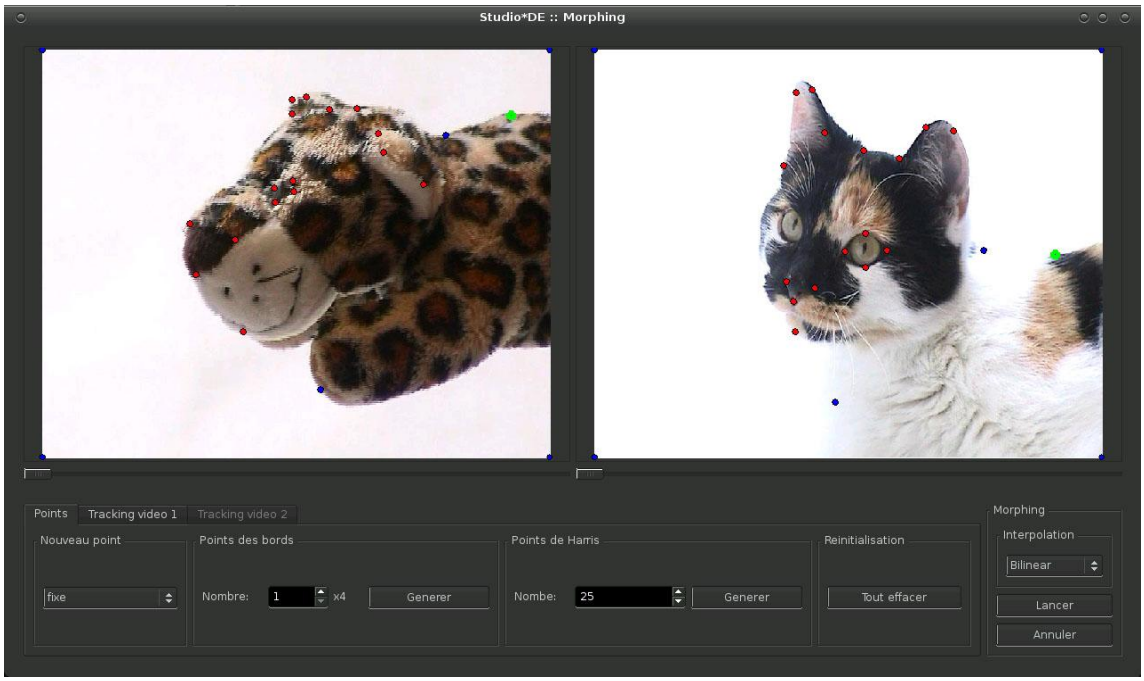

Fig. 24 : Capture d'écran de l'interface du Morphing, avec quelques points cliqués. On voit un couple de points correspondants en vert, deux points fixes en bleu et quatre points sur les bords

B) PREVISUALISATION

Les deux widgets (composants Qt) principaux, dans la partie supérieure, permettent d'abord la prévisualisation des deux médias sur lesquels on travaille. Dans le cas de deux vidéos l'utilisateur dispose d'outils de navigation (séparée ou bien synchronisée pour les deux).

C) AJOUT / SUPPRESSION DE POINTS

En outre, l'utilisateur peut directement sur la vidéo ajouter ou supprimer les points clés du moprhing via un double clic gauche ou droit respectivement. Les points sont ajoutés en parallèle dans les deux fenêtres, à la même position. Ensuite ils peuvent être déplacés indépendamment, le lien entre eux étant toujours conservé et mis en évidence graphiquement. Notons que pour une plus grande précision leur position peut être ajustée au clavier.

D) CREATION AUTOMATIQUE

En plus de la création manuelle de points, l'utilisateur peut se servir de l'algorithme fourni par le programme qui détecte les points de Harris sur l'image. Ce n'est pas une solution parfaitement automatisée car l'algorithme n'est appliqué que sur une des images. Les points trouvés sont ensuite reportés avec les mêmes coordonnées sur la deuxième image et c'est à l'utilisateur de les placer correctement en fonction des correspondances et les effets qu'il souhaite créer.

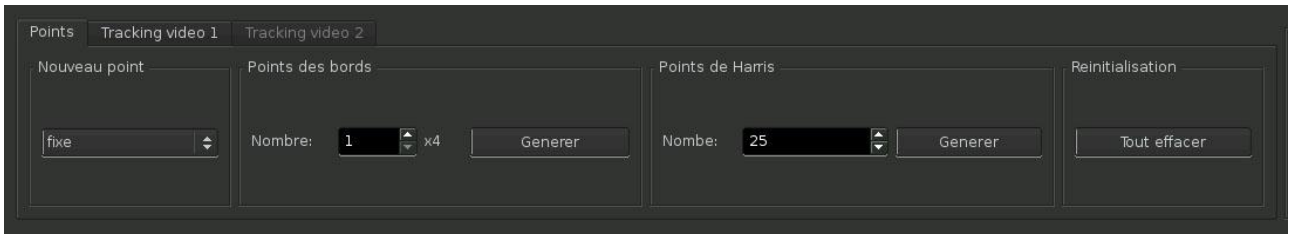

Fig. 25 : Onglet de contrôle et création automatique de points

Par ailleurs dans l'onglet « Points » il peut choisir le type de point qu'il veut créer en cliquant pour indiquer si celui-ci va être soumis au tracking ou non. Il peut également modifier le nombre de points sur les bords des deux images, qui sont indispensables pour la réussite du morphing.

E) REGLAGES DU TRACKING

Par défaut un seul des onglets est activé pour éviter des réglages en double à celui qui veut les mêmes paramètres pour les deux trackings. C'est le cas en général, étant donné que pour un morphing réussi il faut que les deux vidéos ne soient pas très différentes, notamment au niveau de la vitesse et de l'échelle des objets morphés.

Cependant, il suffit d'un clic droit pour activer le deuxième onglet et définir des réglages spécifiques pour le deuxième tracking. Les options disponibles concernent premièrement les types d'algorithmes à utiliser. Comme nous l'avons mentionné précédemment, un seul de ces algorithmes fonctionne de manière suffisamment rapide, mais il est toujours intéressant de pouvoir faire des tests de comparaison. Ensuite l'utilisateur peut régler la taille du bloc de recherche (appelé « Pattern » ou « Motif »), directement en pixels ou bien en pourcentage de l'image. Il peut également définir le rayon de la recherche (taille de la zone de recherche) qui, pour

plus de clarté, est définie en fonction de la taille du bloc. Enfin, le dernier groupe de réglages concernent la gestion d'erreur – son type, notamment si elle est gérée automatiquement, si l'utilisateur souhaite prendre le contrôle total et vérifier chaque image ou bien il veut la désactiver complètement pour un rendu plus rapide et plus autonome. Il peut également spécifier comment est déterminé le « seuil d'erreur » concernant le déplacement des blocs. Il s'agit de spécifier une distance en pixels ou bien de choisir la détermination automatique de celle-ci grâce au système de moyenne construite progressivement par le programme.

Fig. 26 : Onglet de contrôle du tracking. On peut activer le deuxième onglet de réglages par un clic droit

F) INTERFACE DE LA GESTION D'ERREUR DU TRACKING

Basée sur les mêmes classes d'affichage que l'interface du Morphing, c'est une interface très simple qui affiche l'image courante et l'image précédente avec des flèches de navigation entre les deux. Dessus sont mis en évidence les points considérés comme perdus, tandis que les autres sont simplement affichés. L'utilisateur a la possibilité de réajuster leur position, mais sans les supprimer ni en ajouter. Il valide directement ou après correction pour confirmer les changements et relancer le travail de l'interface du Morphing, qui est mise en pause pendant l'intervention.

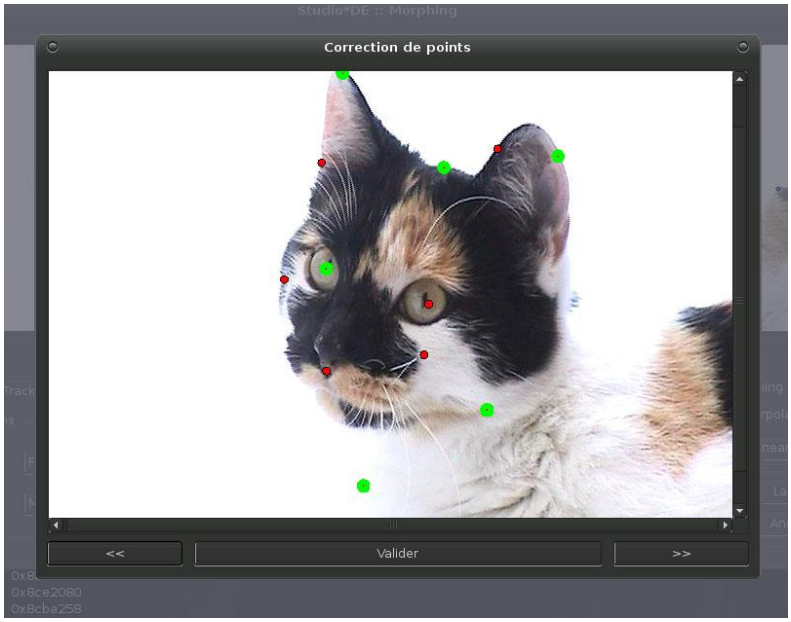

Fig. 27 : Interface de la gestion d'erreur. On voit l'image et les points qui posent problème, on peut les corriger ou valider directement. L'image précédente est disponible via les boutons de navigation pour servir de référence

PROPOSITION DE METHODES DE RENDU

A) PRESENTATION DE LA METHODE DE RENDU OPENGL

A chaque fois que l'utilisateur se déplace sur le scénario, il souhaite avoir un aperçu de ce qu'il fait. De plus il veut pouvoir visualiser le montage qu'il réalise par l'intermédiaire du bouton « Lecture ». Il faut donc mettre en place une stratégie de rendu graphique qui soit intéressante du point de vue qualitatif (rendu rapide et attrayant).

Une première solution brutale consisterait à disposer d'une image de rendu qu'on modifierait à chaque fois que l'on change d'image dans le temps (c'est-à-dire 25 fois par seconde). Pour chaque seconde, nous parcourrons la liste des pistes de la séquence courante (de la piste la plus basse à la plus haute), récupérons pour chacune d'entre elle l'image de la vidéo ou de l'image associée, et l'ajoutons à l'image de rendu. Pour améliorer cela, nous pourrions tester les zones visibles (donc à aller récupérer) ou non. Ensuite nous afficherions cette image à l'écran.

Cette méthode possède de nombreuses limites. La première est qu'elle est assez lourde à gérer, notamment vu la fréquence de rafraîchissement. Ensuite, on ne dispose d'aucune méthode simple pour gérer des effets sur les images.

Une seconde option serait d'utiliser la librairie graphique OpenGL comme moteur de rendu. Son avantage est très net : l'accélération matérielle, dans le cas où elle est supportée par la carte graphique, permet d'augmenter rapidement les performances de rendu. Couplée à l'OpenGL Utility Library (ou GLU), on dispose alors de très nombreuses fonctionnalités. Enfin, OpenGL permet de travailler dans un espace 3D. Cela est très intéressant pour nous, pour l'intégration de plugins de manipulation 3D (de type « perspective » etc.).

Le rendu se ferait alors en utilisant des plans texturés disposés dans un espace 3D. On offre alors à l'utilisateur et au développeur un moyen d'exploiter au maximum les fonctionnalités de la librairie graphique, et de gérer très facilement des effets de type transparence. Ainsi le rendu OpenGL nous permet manipulation plus simple et efficace de l'image (car la manipulation du plan 3D sur lequel elle est plaquée est simple).

Enfin, OpenGL permet de faire du rendu off-screen (on utilise une méthode de rendu RENDER_TO_TEXTURE qui permet de manipuler directement notre image tampon de texture et de l'enregistrer ensuite). On peut de plus exporter directement notre texture vers une sortie écran ou fichier.

Notons simplement une contrainte à cette méthode. Le rendu OpenGL d'une image « normale », c'est-à-dire sans aucun effet, passe par une étape de conversion de l'image source vers le tampon texture, puis une nouvelle vers le fichier. Ainsi, l'image qu'on voudrait simplement copier depuis la source vers la destination, est traitée à plusieurs reprises. Non seulement le temps de traitement est plus long, mais l'image subit des étapes de traitement inutiles et qui peuvent la dégrader.

B) NOTRE IMPLEMENTATION

Pour l'application Studio*DE, nous avons décidé de mettre en place les bases pour le rendu OpenGL. Cela consiste à créer le contexte OpenGL et à rendre une piste de la même manière que vue précédemment.

Cependant nous ne mettons pas à disposition pour le moment d'outils de manipulation dans l'espace 3D des pistes. Cela fera l'objet d'un développement futur.

ANALYSE TECHNIQUE & ARCHITECTURE LOGICIELLE

Nous alors présenter ici les grands principes de fonctionnement de notre application d'un point de vue du développeur. Nous parlerons alors de concepts liés à la programmation objet. Pour les lecteurs peu familiers avec elle, nous présenterons succinctement les grands principes qui lui sont liés.

La programmation orientée objet (indifféremment appelée POO) désigne un langage dans lequel on privilégie l'association de données et méthodes qui agissent sur ces données au sein d'une même entité : la classe. Le concept d'encapsulation résume la philosophie de la POO en cela qu'il désigne qu'on ne peut accéder aux données directement, il faut en effet passer par les méthodes de l'objet qui représentent une interface vers ces données. Un objet est une instance de classe. Une classe peut hériter d'une autre classe, cela permet de définir une classe qui dispose des mêmes méthodes et données de la classe dite « mère » et de ses données et méthodes propres. On peut alors s'appuyer sur l'existant et créer des types « spécialisés » sans devoir redéfinir à chaque fois toutes les fonctionnalités d'un objet.

LES CLASSES

Le schéma présenté ci-après nous propose une vue d'ensemble des classes du projet, c'est-à-dire des entités dont nous avons eu besoin pour mener à bien la réalisation de l'application. Ce schéma sera détaillé et chaque cluster sera analysé en détail dans cette partie.

On peut déjà distinguer plusieurs grandes parties.

- La partie de gestion des données du projet. C'est celle qui réunit les classes qui gèrent les possibilités offertes par l'application, à savoir la gestion des séquences, la description des médias etc.
- Une seconde partie concerne les plugins, c'est-à-dire les fonctionnalités qui ne sont pas présentes de base dans l'application, mais qu'on ajoute au chargement du programme.
- Enfin, une dernière partie concerne l'interface graphique, très liée aux données.

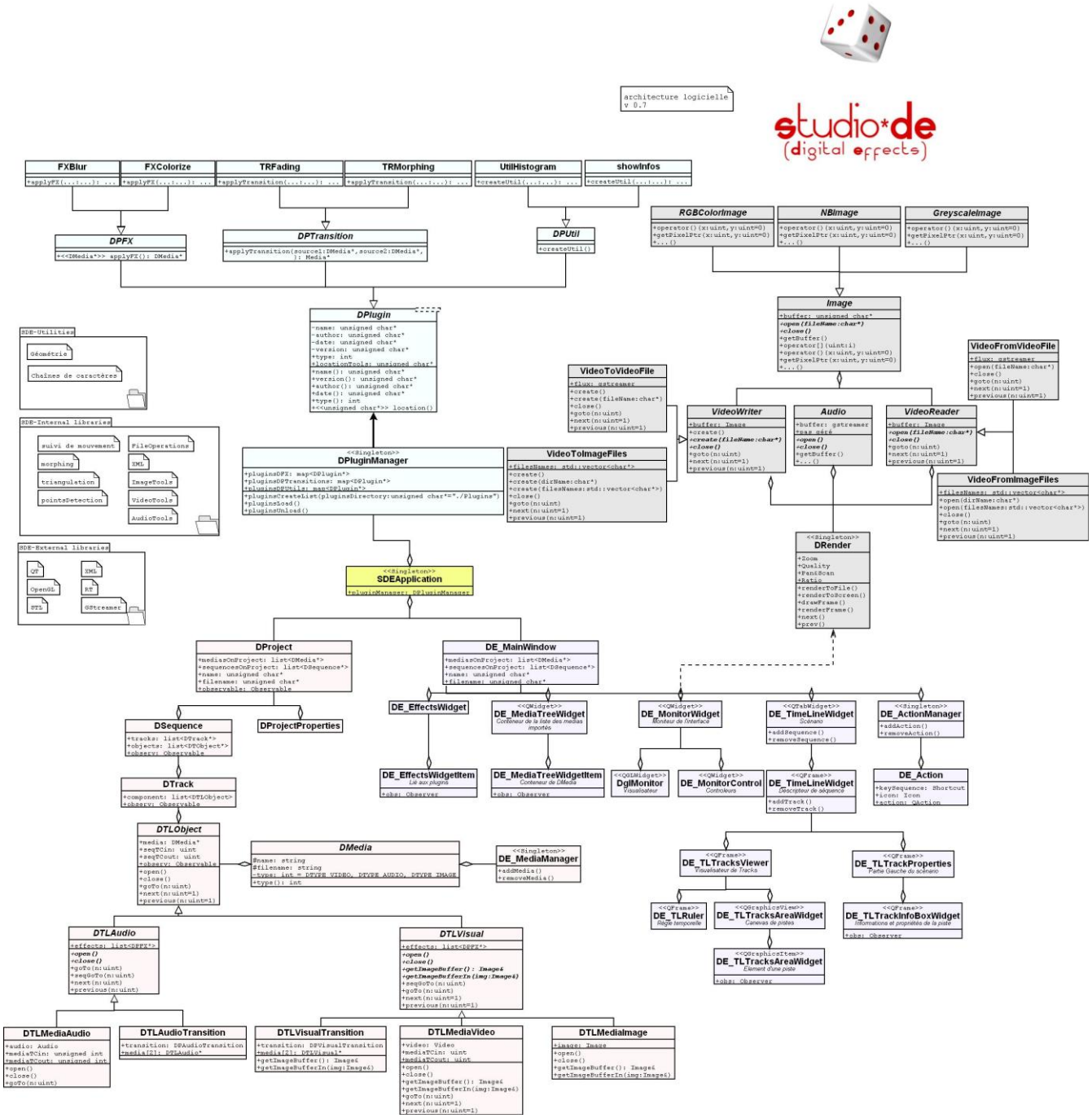

Fig. 28 : Schéma global des classes de l'application

A) LES DONNEES DE PROJET

Le projet est représenté par des données et une interface qui permet d'y accéder et de les modifier. Ces données sont représentées par le schéma ci-après.

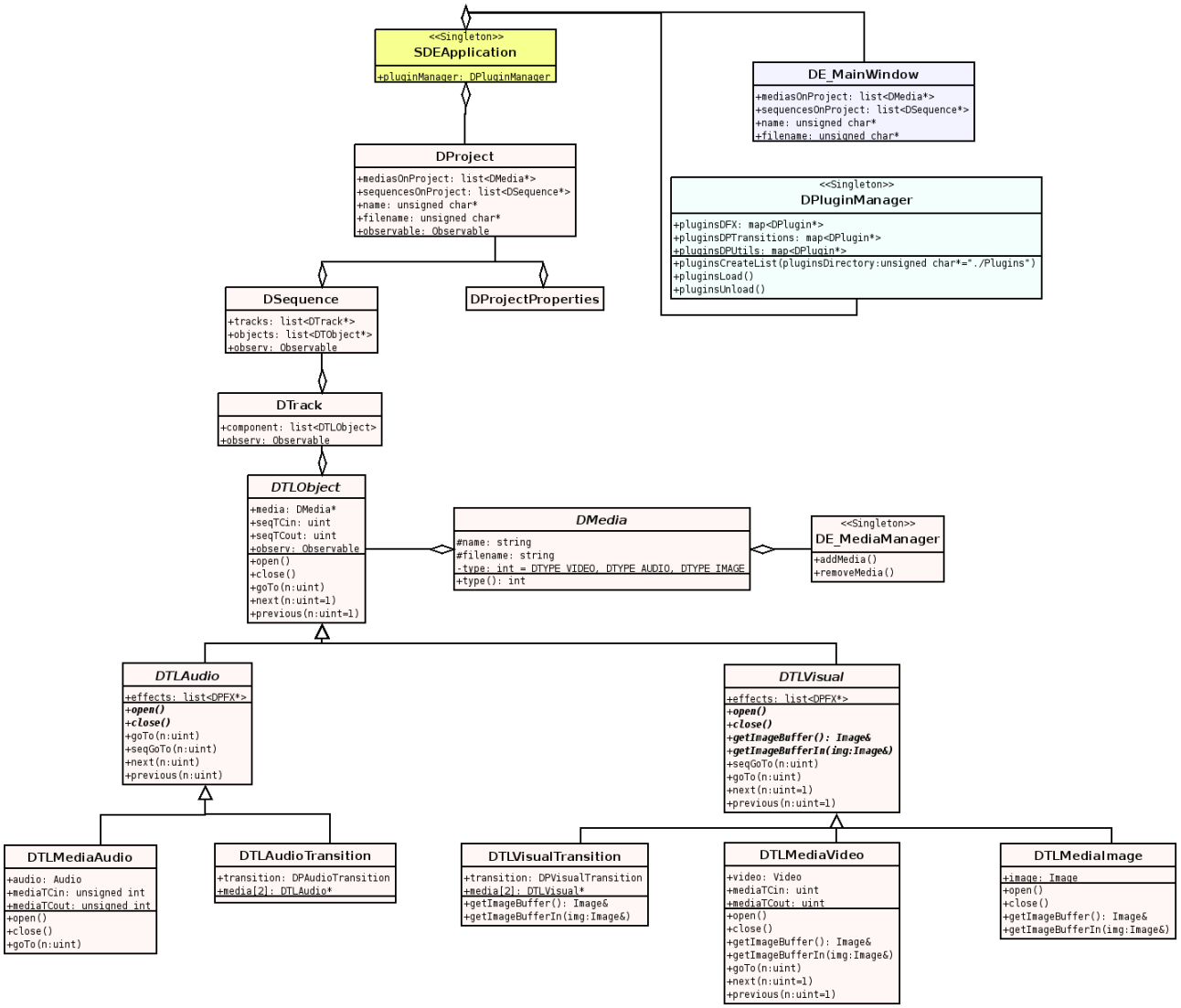

Fig. 29 : Schéma des données de l'application

La classe DE_Application est la classe principale du projet puisque c'est à partir d'elle que se construit l'application. Son rôle est principalement d'avoir en son sein un projet (DE_Project) et une interface (DE_MainWindow).

La classe DE_Project décrit les données et méthodes d'un projet (sa création, sa sauvegarde). Elles font notamment référence à la constitution d'un projet : un ensemble de séquences (DE_Sequence), un nom, un emplacement de fichier etc.

Ensuite on entre dans la constitution d'un projet vidéo. La première étape concerne la création d'une séquence. Chaque séquence peut contenir une ou plusieurs tracks (ou pistes) vidéo

et audio. Chaque piste contient une suite de médias (pour une piste visuelle il peut s'agir d'une image fixe ou d'une vidéo) glissé depuis la liste des médias précédemment citée.

Ces données sont interfacées à l'utilisateur à l'aide d'une fenêtre réalisée avec le framework Qt.

B) L'APPLICATION ET LA FENETRE PRINCIPALE

L'application est lancée grâce au fichier main.cpp (appelé par la commande **./studiode**) qui contient la méthode principale du programme. Son contenu est très court puisqu'elle demande simplement l'exécution de l'instance de De_Application. Celle-ci va charger l'interface et les données jugées importantes, disponibles tout au long du programme. Le démarrage de l'application se fait selon le schéma présenté dans la partie suivante (Exécution du programme).

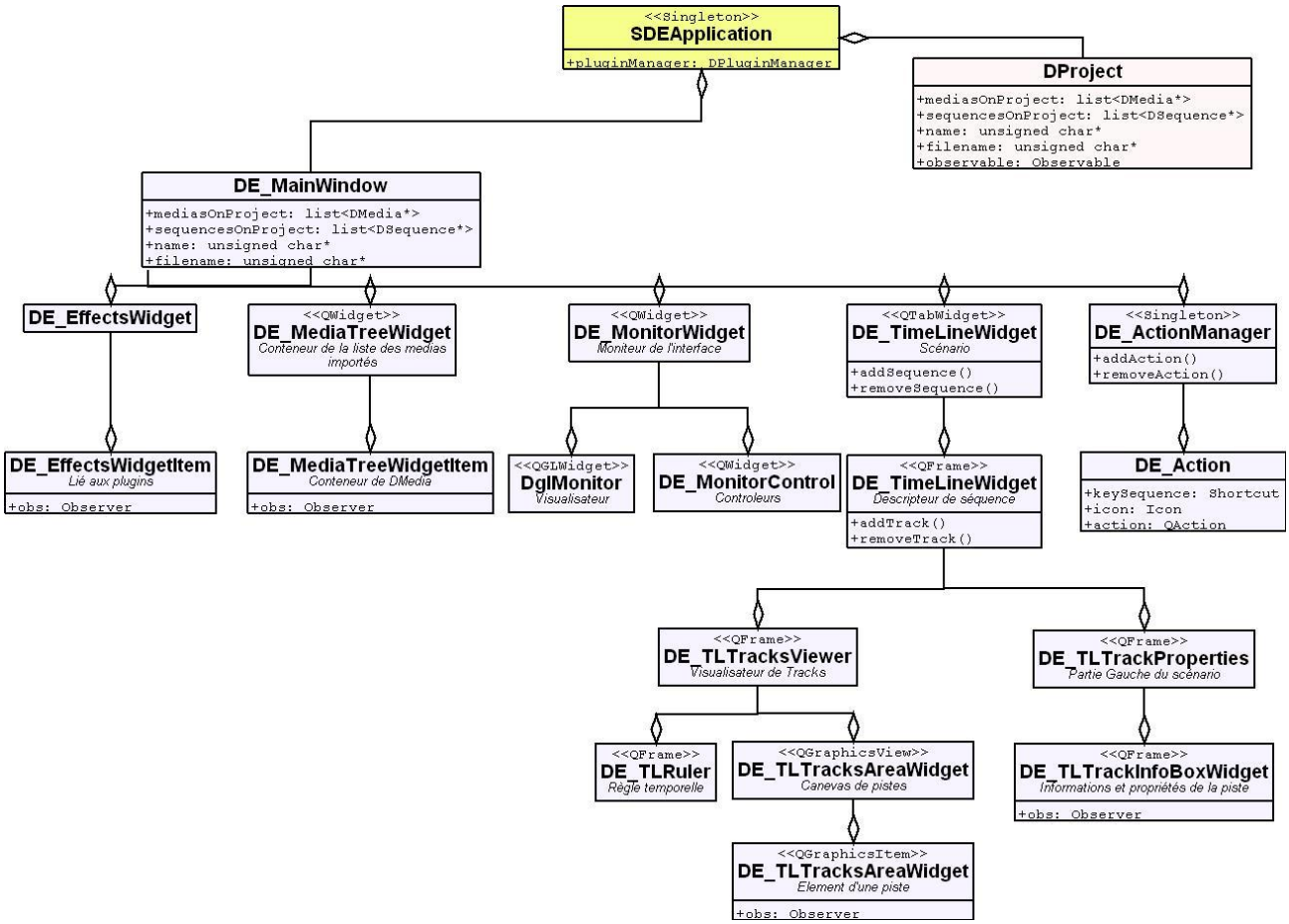

Fig. 30 : Les classes de l'interface principale

La classe DE_MainWindow décrit l'interface graphique du programme. C'est elle qui, héritant de Qt::QMainWindow, permet l'affichage à l'écran d'une fenêtre de visualisation du projet. Elle construit son interface en fonction de ce qui lui a été ordonné par nos soins (emplacement des widgets [principale unité graphique de Qt définissant quelque chose à l'écran]). C'est également à elle que revient la tâche de placer les menus et d'y associer les actions. Nous la lions au projet pour pouvoir accéder rapidement aux données vues précédemment. Le principe est

de distinguer l'interface et les données. Les données ne doivent pas connaître le système de l'interface, ni la modifier elle-même (cela pour ne pas se contraindre à une seule interface, et pour sécuriser les données). Nous verrons par la suite quels mécanismes ont été mis en place pour permettre cette indépendance des données vis-à-vis de l'interface.

C) LES CLASSES DU TRACKING

Ce sont quatre classes de base dont l'enchainement de l'interaction effectue le suivi de mouvement. Nous pouvons approximer cette interaction par le schéma suivant :

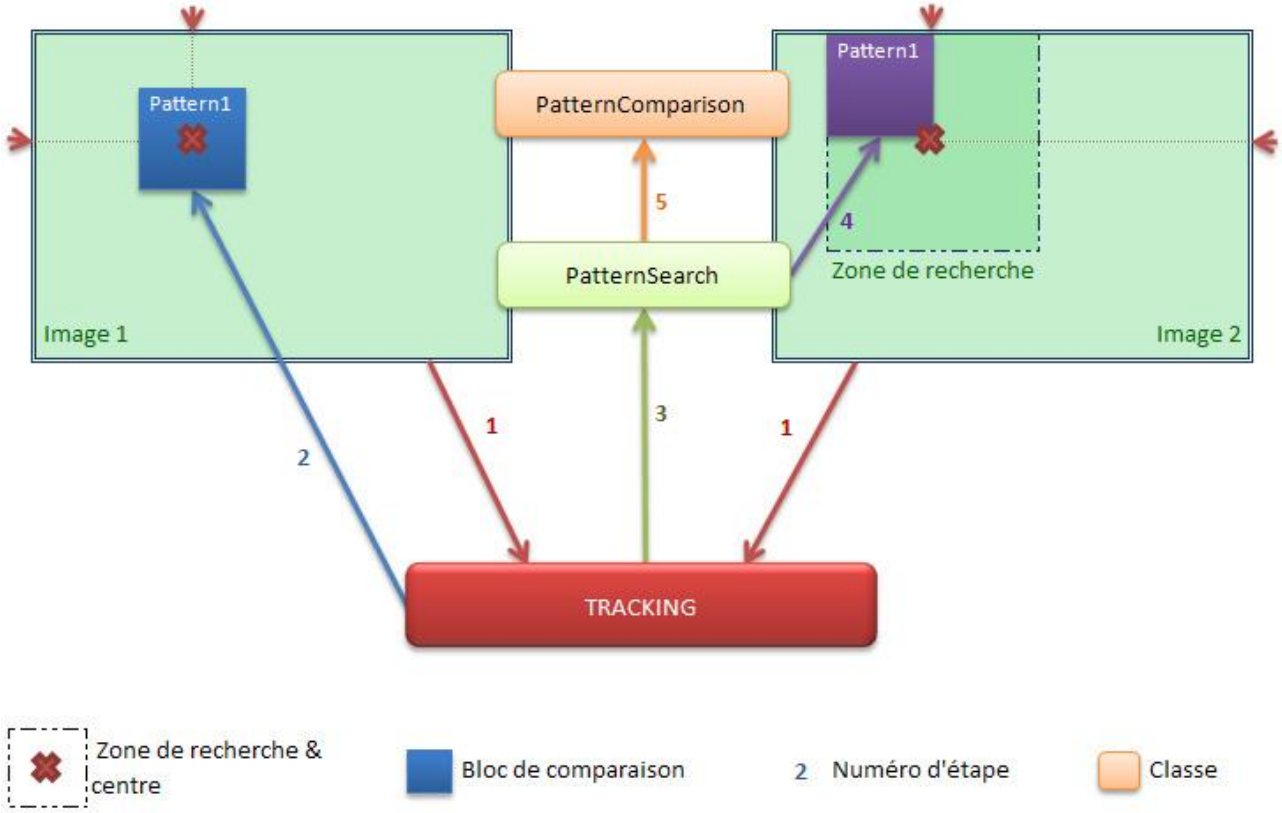

Fig. 31 : Aperçu schématique du fonctionnement du Tracking

La classe Tracking est celle qui fait le suivi final et qui appelle toutes les autres.

- Le **Tracking** prend d'abord deux images **(1)**.
- Puis elle crée une instance de la classe **Pattern** basée sur la première image (2).
- Elle appelle ensuite le **PatternSearch** en lui passant ce Pattern ainsi que la deuxième image (3).
- PatternSearch de son côté crée le deuxième **Pattern** à partir de l'image reçue (4).
- Finalement il utilise les fonctions de **PatternComparison** pour comparer les deux Patterns (5).

L'information remonte ensuite pour être traitée par le Tracking.

D) Les classes du morphing

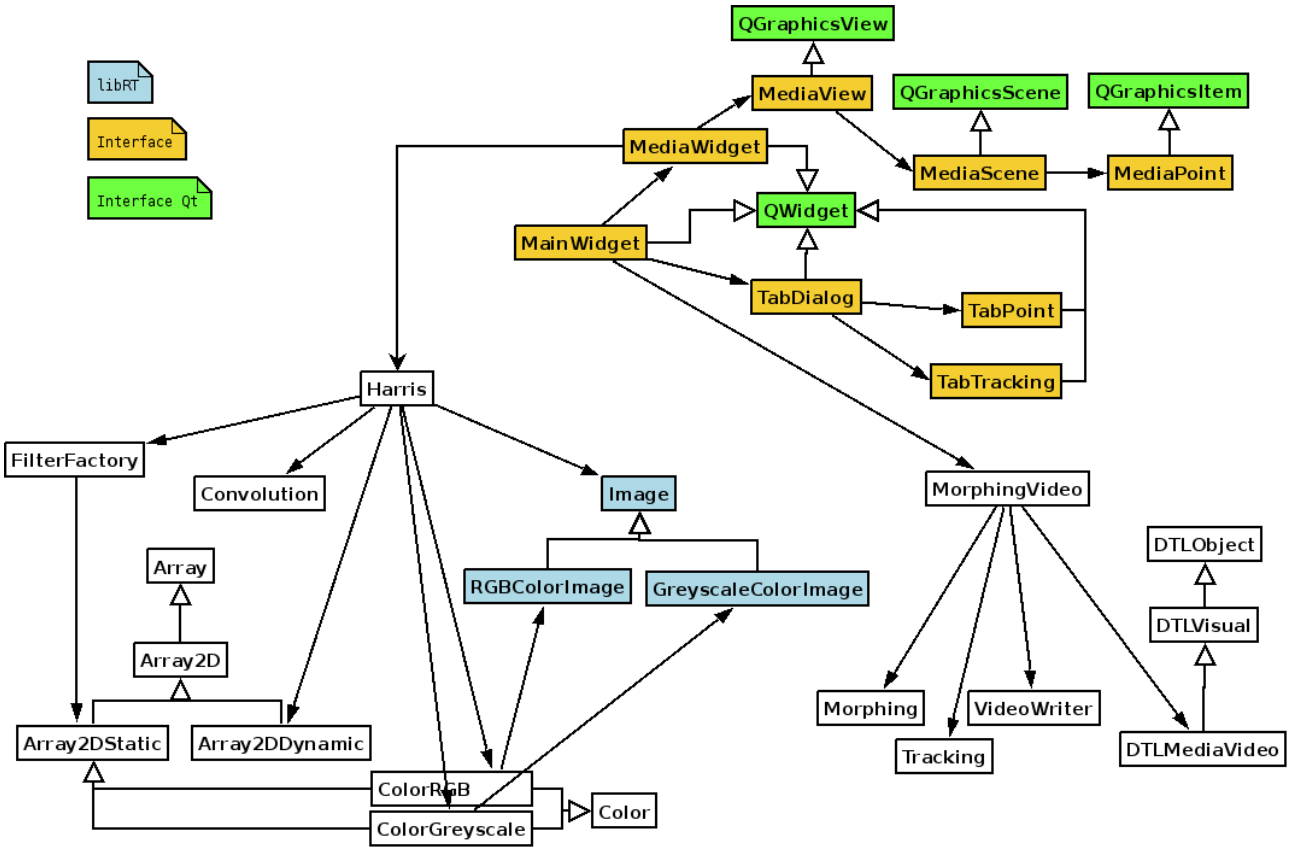

Fig. 32 : Les classes du plugin morphing et de son interface

On voit sur ce schéma, l'organisation schématique des classes de l'effet de morphing entre deux vidéos. Il gère d'un coté l'appel à la classe Harris pour générer les points clefs à partir d'une image, et de l'autre, l'appel à la classe MorphingVideo pour la paramétrer depuis l'interface et lancer le morphing.

E) ORGANISATION DES PLUGINS

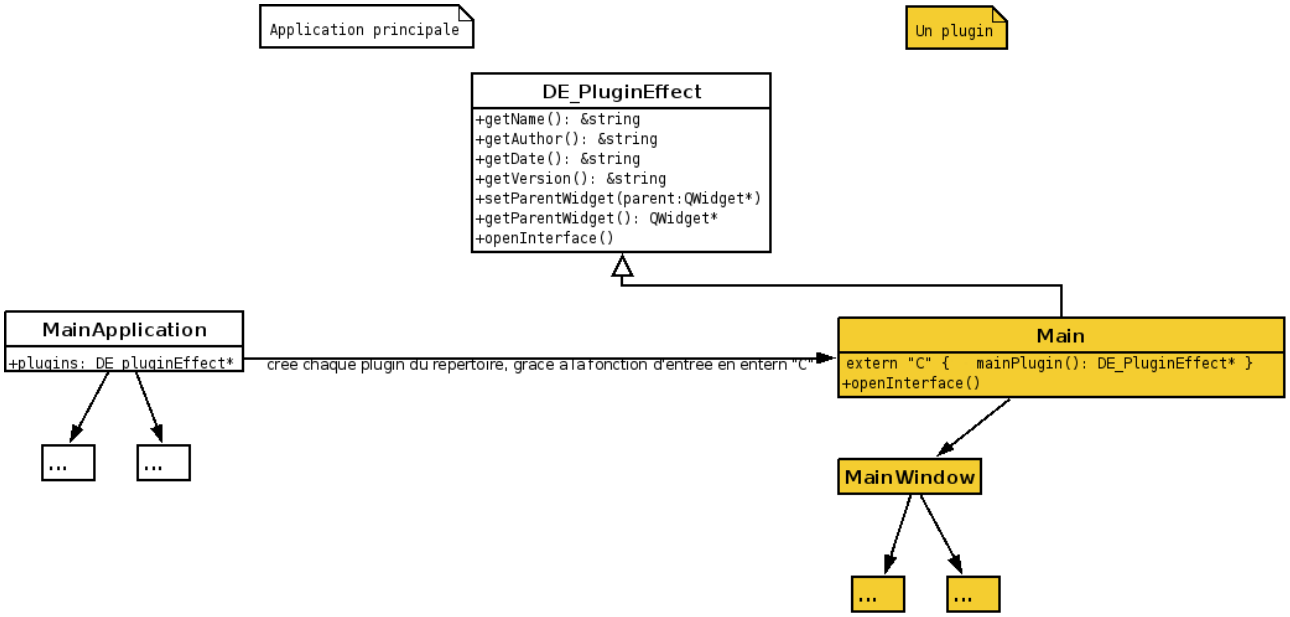

Fig. 33 : Mécanisme de liaison de l'interface principale avec les plugins

Comme l'on voit sur ce schéma les plugins on accès à une partie des classes de l'application (les classes qui sont dans le dossier « common »). La classe de base du plugin appelé ici « Main » (comme dans l'exemple réalisé) hérite de la classe générique des plugins « DE_PluginEffect » qui définit les fonctions de bases des plugins.

Le programme principal appelé ici « MainApplication » crée une liste d'objets « DE_PluginEffect » à partir d'un *#define* définit dans le fichier cpp principal de chaque plugin. *Ce #define* déclare une fonction en extern « C » permettant de créer l'objet principal du plugin (ici « Main »).

EXECUTION DU PROGRAMME ET POINTS PARTICULIERS

A) UN MOTIF DE CONCEPTION PARTICULIER : MVC

C'est une représentation d'architecture pour le développement d'applications qui permet de séparer le modèles de données (à savoir, pour notre projet, les classes de type De_Project, De_Sequence, ...), l'interface utilisateur et la logique de contrôle (commandes exécutées pour agir sur les données, par exemple l'application d'un effet). L'intérêt est non négligeable. On s'affranchit, en utilisant cette méthode de développement, de mélanger tout le code de l'application dans quelques gros fichiers. C'est-à-dire qu'on sépare les données, leur traitement et la présentation. En fait, notre application utilise plutôt un motif de type Model / View-Controller. En effet, c'est généralement l'interface qui va appliquer les transformations aux données et on ne passe pas par un contrôleur (gestion et synchronisation des événements). Ce détail a son importance. Nous avions engagé notre méthode de programmation ainsi dès le départ, ce qui, nous nous en sommes aperçu, peut poser problème et explique en partie pourquoi nous n'avons pas implémenté les méthodes « Undo /Redo». Nous pensons, pour le développement futur de l'application, y remédier. En effet, on a actuellement (nous le verrons dans le point suivant) des entités spécialisées dans la gestion des médias importés par exemple, et cela pour tout le projet (accessible à toutes les classes) qu'on nomme managers (gestionnaires). Ces managers permettent une gestion globale mais sont spécifiques à un point. Cependant, c'est bien le contrôleur qui permettrait de centraliser et distribuer les informations sur les actions à entreprendre entre l'interface et les données. Le fait de disposer d'une entité centrale nous aurait autorisé à contrôler ces actions, et à les stocker dans une liste qui nous aurait permis alors de proposer des fonctionnalités supplémentaires.

Actuellement donc, nous adoptons le modèle proposé par le schéma suivant :

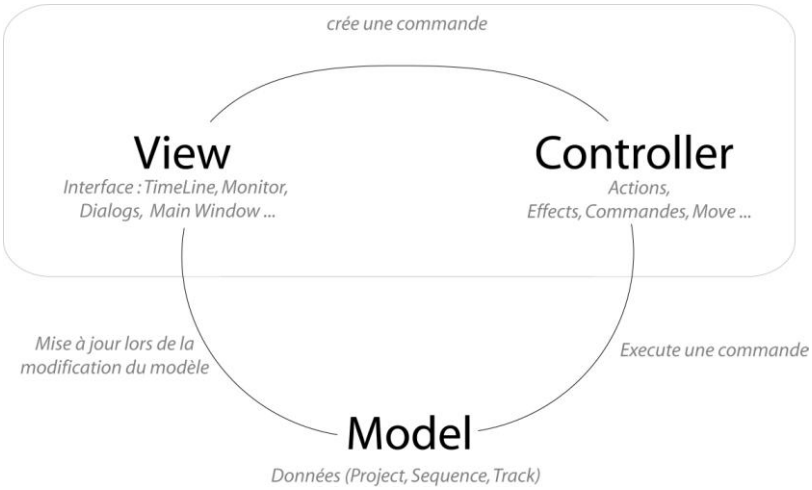

Fig. 34 : Vue schématique du model / view controller

Cela nous permet par contre d'avoir une architecture claire du programme, ce qui permet une maintenance plus facile, mais surtout les données sont totalement indépendantes de l'interface. C'est-à-dire qu'il nous sera facilement possible de passer du framework Qt à un autre sans modifier les fichiers de données, mais simplement ceux de l'interface.

B) LES AUTRES MOTIFS DE CONCEPTION

Au cours du développement nous nous sommes vite aperçu que l'ampleur d'un projet tel que le notre nécessiterait des entités de centralisation et de contrôle sans lesquelles il aurait été extrêmement compliqué d'aboutir à un résultat sûr et efficace. On appelle ces entités des motifs de conception (ou design patterns). Ce sont des méthodes développement qui permettent d'obtenir d'un objet qu'il contrôle ou gère d'autres objets.

- **Le singleton** : c'est une entité spécifique qui n'existe qu'une et une seule fois dans le programme. C'est-à-dire qu'elle peut être appelée de n'importe où, elle sera unique, et ne sera pas ré-instanciée à chaque fois. C'est important dans une application comme la notre de disposer de cette entité car elle permet de centraliser des données nécessaires à de nombreuses autres entités (car elles sont accessibles partout), et surtout de ne les avoir en mémoire qu'une seule fois.

  Dans notre application, il existe plusieurs singletons, notamment De_Application puisque l'application doit être unique (on ne peut pas lancer deux fois une application à partir de l'application), ou les managers (par exemple De_MediaManager)

- **L'usine (factory)** : ce motif permet d'utiliser des méthodes sans avoir à instancier un objet. Les méthodes sont applicables à tous les objets. C'est une grande aide au développement car il nous est possible d'ajouter des méthodes (par exemple de gestion de fichiers, ou de tableaux) sans avoir à recréer autour une architecture complexe.

- **Les managers** sont des singletons particuliers qui gèrent un point précis. Par exemple, le DE_MediaManager s'occupe de centraliser les médias importés au sein d'une même entité. Il en est de même pour le De_ActionManager qui va centraliser toutes les actions (du point de vue des menus) pour que, si l'on souhaite créer deux boutons dans deux espaces différents de l'interface, mais que ces deux boutons aient un comportement identique, alors on ne duplique pas l'information de l'action d'un bouton deux fois. Cela permet surtout, lorsqu'on veut modifier l'action de ces boutons, de ne le faire qu'une fois.

- **Les observateurs (observers)** : Cela a été le point certainement le moins aisé à comprendre, bien que la mise en œuvre soit très simple. Son rôle est primordial puisqu'il est au centre du modèle MVC. Il permet à une entité de signaler un changement de son état à d'autres entités qui le désirent. C'est très pratique par exemple pour la communication entre les données et l'interface. L'interface s'inscrit auprès d'une entité de données (sans que celle-ci sache qui elle est), et lui indique ainsi, lorsque son état change, qu'il faut qu'elle se mette à jour. Ainsi, un observateur se lie à un observable, qui, lorsqu'il sera modifié, fera appel à une fonction de mise-à-jour qui fera à son tour appel aux fonctions de mise-à-jour des observateurs.

Ci-après on montre un schéma non exhaustif des relations d'observateurs/observables qui ont lieu au sein de l'application.

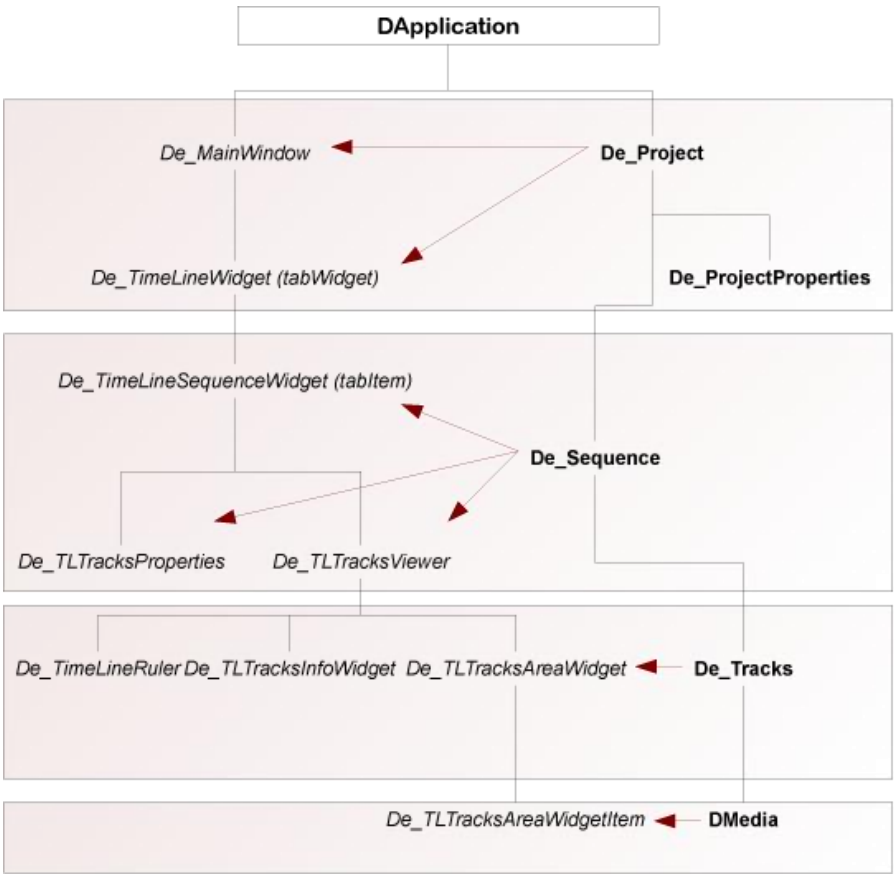

Fig. 35 : Le mécanisme d'utilisation d'un observateur

L'utilisation de ces motifs est très pratique, mais elle ne se fait pas sans la nécessaire multiplication des fichiers. Il est d'ailleurs intéressant de voir que la séparation en MVC accroît considérablement le nombre de fichiers sources. Il faut donc s'imposer une structure de nommage et de classement très stricte, sans quoi il devient compliqué de s'y retrouver.

C) DEMARRAGE DU PROGRAMME

Lorsque l'utilisateur lance le programme, les actions suivantes sont effectuées par l'entité DE_Application :

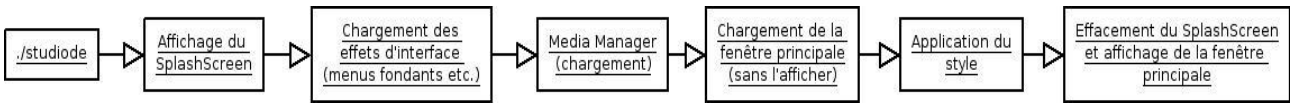

Fig. 36 : Pipeline du chargement du programme

On pré-charge donc les outils dont on aura besoin. Pendant ce temps de chargement, on affiche à l'écran un spash-screen qui fait état de l'avancement du chargement. Pour notre projet, et à notre stade (peu de plugins, peu d'entités à pré-charger), cela est assez rapide. Mais lorsque l'application prendra plus d'ampleur, que les plugins se multiplieront, le splash-screen trouvera

tout son intérêt. En effet, un utilisateur non averti pourra penser que l'application ne marche pas si elle est démarrée mais que rien n'apparaît à l'écran. Il est dérouté et peut vouloir quitter le programme de manière brutale.

D) ÉVENEMENTS ET DRAG' N' DROP

La gestion des événements ainsi que le système de drag 'n' drop utilisé sont mis en place par la plateforme Qt. Il faut alors implémenter toutes les méthodes de gestion de ces événements. C'est une partie plutôt intéressante à développer. On a alors une certaine conscience des réelles possibilités proposées à l'utilisateur. On contrôle ce qui peut être ou non fait. Cependant cela requiert une analyse pour ne pas se perdre dans les possibilités offertes par chaque événement. Il faut ainsi toujours penser l'interaction dans un domaine plus global que l'entité sur laquelle elle agit. Par exemple il faut penser à la cohérence des interactions les unes par rapport aux autres.

Les événements (dont le drag 'n' drop, les clics de souris, la molette) sont gérés par des interfaces (méthodes vides) qu'il faut redéfinir. On a ensuite accès à de nombreuses fonctionnalités implémentées par Qt (accès à la position de la souris à l'écran) qui nous permettent de nous concentrer sur le but de l'événement, et non pas la construction de cet événement.

Pour information, les méthodes utilisées sont décrites ci-dessous :

mousePressEvent(QMouseEvent)*	Est l'interface de gestion de pression sur un bouton de la souris.
mouseDoubleClickEvent(QMouseEvent)*	Est l'interface de gestion de double clic sur un bouton de la souris.
mouseMoveEvent(QMouseEvent)*	Est l'interface de gestion de pression et déplacement d'un bouton de la souris.
mouseReleaseEvent(QMouseEvent)*	Est l'interface de gestion de relâchement d'un bouton de la souris.
dragEnterEvent()	Est l'interface de reconnaissance d'un drag (déplacement) entrant sur un objet (widget)
dragMoveEvent()	Est l'interface de reconnaissance d'un drag (déplacement) sur un objet (widget)
dropEvent()	Est l'interface de reconnaissance d'un drop (relâchement) d'un objet sur un autre objet (widget)

Qt nous autorise également à gérer les interactions avec le clavier (par exemple appuyer sur la barre d'espace pour mettre en pause ou lecture la vidéo).

E) STRUCTURE DE LA SAUVEGARDE

La gestion de la sauvegarde n'a pas été entièrement implémentée. Nous présentons cependant la structure de la sauvegarde d'un projet sur Studio*DE.

Nous utilisons le langage générique XML (eXtensible Markup Language) qui permet de facilement récupérer un arbre de données décrivant un projet tel que le notre. Cet arbre est construit lors de la sauvegarde et enregistré dans un fichier décrit ci-dessous :

<?xml version= « 1.0 » encoding= « UTF-8 »>	Définition de la version XML
<Project>	Un projet est décrit par ...
<Properties> ... </Properties>	- des propriétés (nom, auteur, ..), classe De_ProjectProperties
<MediaList> *<MediaListElement>...<MediaListElement>* *<MediaList>*	- une liste de médias importés (DE_MediaTree)
<Sequence name= ...>	- une liste de séquence
<Track>	- par séquence, une liste de pistes et leur contenu
<MediaOnTL>	- par piste ses attributs et les éléments qui la constitue
<FX> ... </FX>	- par élément les effets appliqués (au chargement il faut donc vérifier que les effets existent)
</MediaOnTL> ... </Track>	
</Sequence>	
</Project>	

ORGANISATION DES FICHIERS

Voici la liste des dossiers, l'explication de leur organisation et une explication sur les fichiers qu'ils contiennent.

- **doc_doxygen**

  Contient la documentation du code générée par Doxygen.

  - **config**

    Contient les fichiers pour les parties rédigées de Doxygen (pages d'index, texte de licence, documentation des namespaces).

- **src**

  Contient toutes les sources de l'application. Le code de l'application est séparé en 3 grandes parties qui représentent :

  - une partie « application » qui contient l'application principale avec toute la gestion de l'interface du logiciel
  - une partie « common » qui fonctionne un peu comme une librairie et dont les classes peuvent être utilisées depuis les plugins
  - une partie « plugin » qui contient une liste de dossiers de plugins (chacun de ces dossiers correspondant à un plugin)

  Les *#include* des fichiers sont en référencement absolu dans toute l'application, à partir des répertoires « de ». Ainsi le code peut être déplacé entre les 3 parties sans poser de problèmes de liaison d'entêtes (aucun changement n'est nécessaire dans le code).

  - **application**

    Contient toutes les sources propres à l'application. Ces classes ne sont pas accessibles aux plugins.

    - **de**

      Répertoire "de" servant de racine pour les *#include*.

      - **conf**

        Contient les fichiers de configuration du programme. On y trouve actuellement les données par défaut de certains éléments (taille du scénario, taux d'images par seconde etc.)

      - **core**

        Contient les fichiers relatifs aux contrôles (du modèle Model-View-Controller) dédiés au programme. On y trouve les fichiers d'actions spécifiques à l'interface entre notre programme et Qt.

      - **edition**

Contient les fichiers relatifs aux données du scénario.

- **gui**

  Répertoire contenant les fichiers ayant rapport à l'interface graphique.

  - **styles**

    Styles applicables à l'interface (changement de formes de boutons etc.). Ce répertoire pourra contenir à l'avenir les différentes déclinaisons graphiques de l'interface (choix à la volée à partir du programme)

  - **uic**

    Contient les fichiers spécifiquement créés par QtDesigner (assistant création d'interface). Actuellement il n'est pas utilisé pour l'application. Mais c'est une aide certaine pour le développement futur d'éléments graphiques.

  - **utils**

    Contient des fichiers génériques d'éléments graphiques.

- **common**

  Contient toutes les classes génériques, ces classes peuvent être utilisées aussi bien par l'application que par les plugins.

  - **de**

    Répertoire "de" servant de racine pour les *#include*.

    - **geometry**

      Classes de base pour les opérations géométriques, par exemple calcul de distance.

      - **interpolation**

        Interpolation géométrique, avait été une piste pour optimiser le tracking et serait intéressant pour faire de l'interpolation temporelle en vidéo.

      - **image**

        Classes de base pour gérer les images, comme l'interpolation colorimétrique. Les classes Images ne sont pas ici, car celles que nous utilisons sont celles de la librairie RT.

        - **color**

          Ensemble de classes pour gérer les couleurs d'une image (récupérer un tableau de N composantes d'une image).

        - **file**

          Gérer l'ouverture et l'enregistrement des images en différents formats : jpg, ppm, tga.

- **filter**

  Les filtres d'images (Convolution et ensemble de kernels pour faire des filtrages)

- **pointsOfInterest**

  Filtres de détection de points d'intérêt dans une image, actuellement seulement la détection des points de Harris.

- **math**

  Classes de base de math, actuellement uniquement des classes de matrices.

  - **array**

    Classes de base pour gérer des tableaux statiques ou dynamiques (de la même manière) et pour gérer des tableaux en 2D ou 3D.

- **mathBases**

  Classes de math vraiment basiques, comme les puissances entières, les arrondis, etc.

- **media**

  Classes pour gérer les objets image, vidéo, audio qui sont sur la timeline. Les effets travaillent à partir de ces éléments Media.

- **plugin**

  Classe pour charger les plugins et classe de base des plugins.

- **string**

  Classes de base pour des petits traitements de chaînes de caractères.

- **tracking**

  Le module de tracking ne devrait pas être ici, il devrait être un plugin qui n'est pas un effet en soi mais que les autres plugins peuvent utiliser. Par contre cela nécessite une gestion des plugins beaucoup plus poussée pour gérer les dépendances des plugins.

  - **errorGUI**

    Interface de gestion d'erreur du tracking. Ses classes graphiques (scène et points) sont également utilisées par l'interface du morphing.

- **utils**

  Ensemble de classes génériques :

  - Debug et NoDebug : proposent des *#define* pour simplifier le débogage et pour permettre de compiler en 2 modes, avec ou sans débogage
  - Singleton : classe de base pour simplifier la création de classes en singleton (il faut hériter de cette classe)

- TimeGlobal : pour chronométrer des fonctions, etc.
- FileGlobal : créer facilement des fichiers de log, et d'écrire dedans depuis n'importe quelle fonction
- LimitsOfTypes : pour avoir les valeurs min/max du type utilisé (int, double, etc.) de façon simple notamment pour les classes en templates
- FileTools : traitement de fichiers, par exemple récupérer tous les fichiers d'un répertoire, l'extension d'un fichier, etc.
- Observer :
- Observable :
- etc.

- **video**

  Classes pour gérer la vidéo, soit depuis GStreamer, soit une liste de fichiers images comme une vidéo

- **plugins**

  Chaque sous-répertoire correspond à un plugin particulier.

  - **de**

    Répertoire "de" servant de racine pour les *#include*.

    - **example**

      Plugin d'exemple contenant une interface vide, permettant ainsi à toutes les personnes qui voudraient créer un plugin d'avoir une base pour le développer.

      - **GUI**

        Gestion de l'interface du plugin

    - **morphing**

      Plugin de morphing automatisé entre 2 vidéos

      - **GUI**

        Interface du plugin de morphing

- **tools**

  Les librairies utilisées : actuellement uniquement la libRT

- **pluginsEffects**

  Contient tous les plugins compilés sous forme de librairies dynamiques (.so)

Pour plus d'information sur les classes, les fichiers, l'organisation du code vous pouvez vous reporter à la documentation doxygen, disponible notamment sur le site internet.

CODAGE

Pour mener à bien notre projet, nous avons utilisé plusieurs outils de développement.

- **La STL**, fournie de base avec les implémentations du compilateur, est une librairie très utile car elle offre des outils basiques mais performants (notamment l'utilisation intuitive de tableaux de données) que nous n'avons pas à recoder.

- **Doxygen** nous permet de générer une documentation complète et organisée du code source, monnayant une discipline de codage et une attention toute particulière portée aux commentaires qui doivent expliquer le rôle de chaque entité. Commenter pour Doxygen est long mais le résultat est à la hauteur de nos attentes. La documentation du projet actuel est disponible avec le cd-rom fourni lors de la soutenance, ou bien consultable en ligne sur le site studiode.free.fr.

RESULTATS

MORPHING

Nous avons commencé à implémenter le morphing en se basant sur un espace vectoriel, mais les cas particuliers étant difficiles à gérer nous avons obtenu des résultats très surprenants.

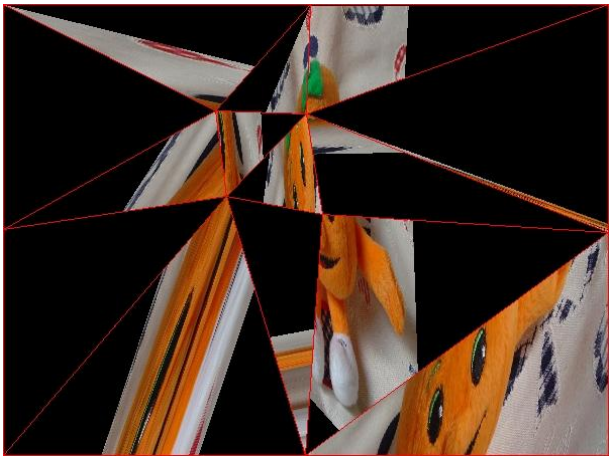

Fig. 37: Premiers tests dans l'espace vectoriel, les paramètres sont mal réglés

Fig. 38 : Premiers résultats basés sur les coefficients des sommets des triangles. Le résultat est bon mais les cas particuliers ne sont pas gérés

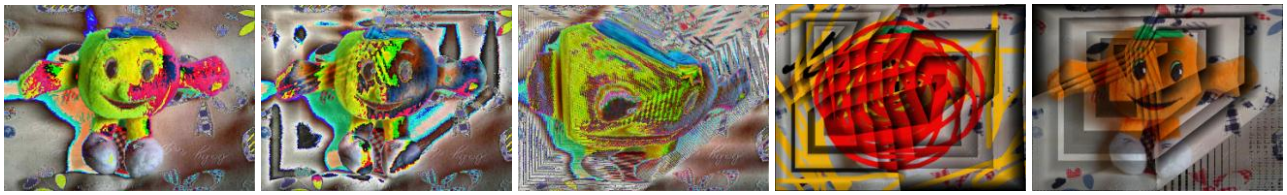

Fig. 39 : Ensemble de résultats erronés plutôt surprenants

Après avoir réussi à implémenter le morphing, nous avons rajouté la gestion de l'interpolation (pour l'instant uniquement *nearest* et *bilinear*).

HARRIS

En ce qui concerne les points de Harris quelques petites erreurs dans le calcul des scores ont montrés que ceux-ci sont beaucoup moins équilibrés dans l'image si l'on ne respecte pas bien la formule de Harris.

Fig. 40 : Points de Harris

Nous obtenons les scores en inversés, c'est à dire, que les meilleurs scores sont négatifs. Mais les façons d'implémenter Harris varient les unes les autres. Comme nous partons d'une matrice pour le filtre de Sobel qui est équilibré (la somme des coefficients de la matrice vaut zéro), ce résultat est peut-être normal.

TRACKING

Les premiers tests du tracking pour notre logiciel ont été effectués entre deux images légèrement différentes, souvent modifiées à la main. Nous exposons ici les premiers résultats de vraies séquences vidéo.

Ce premier couple de blocs suivis est intéressant pour remarquer l'importance de la taille du bloc pour la réussite du tracking. Notons également que ces tests sont effectués sur des séquences fortement échantillonnées (2 images/sec. Approximarivement).

Fig. 41a : Tracking : 2 img/sec avec des blocs de 50x50 pixels

Fig. 41b : Tracking : 2 img/sec avec des blocs de 62x62 pixels

La première suite comporte des blocs de 50x50 pixels. Les couleurs des cadres expriment la vitesse à laquelle le bloc est « perdu ». La deuxième ligne est composée de blocs de 62x62 pixels et on remarque la meilleure qualité du suivi. Nous en avons tiré la conclusion que les blocs plus grands dans ce cas concret étaient plus adaptés à l'échelle de l'objet. Ce qui veut dire qu'ils sont assez grands pour contenir assez d'informations caractéristiques pour une meilleure reconnaissance du motif. Dans le premier suivi ci-dessus l'objet est déjà légèrement rogné au début, ce qui facilite le décalage progressif qu'on observe.

Nous présentons également les résultats des tests après implémentation des différentes fonctions de distorsion. A cette étape nous utilisions déjà des séquences complètes (25 img/sec) et des blocs de comparaison plus petits pour se rapprocher de la réalité de l'utilisation future du logiciel.

Fig. 42a : Tracking à base de Pel Difference Classification (25 img/sec avec des blocs de 14x14 px)

Fig. 42b : Tracking à base de Mean Squared Differences (25 img/sec avec des blocs de 14x14 px)

Fig. 42c : Tracking à base de Mean Absolute Differences (25 img/sec avec des blocs de 14x14 px)

Fig. 42d : Tracking à base de Mean Absolute Differences sur les lignes uniquement (25 img/sec avec des blocs de 14x14 px)

Ces quatre suites présentent un échantillon de nos résultats, à raison d'un bloc sur 5 pour la partie en vert et d'un bloc sur 2-3 pour la partie en jaune-rouge, pour montrer plus en détail comment le motif est perdu.

L'intérêt de cette comparaison est surtout la similarité des résultats. A l'exception du PDC, les trois autres algorithmes montrent une performance infiniment proche, on peut dire sans défaut, car le fait de perdre le motif dans cette séquence est inévitable, vu que le sujet se retourne.

INTERFACE

Voici une capture d'écran qui témoigne de l'avancement de l'interface principale à ce jour.

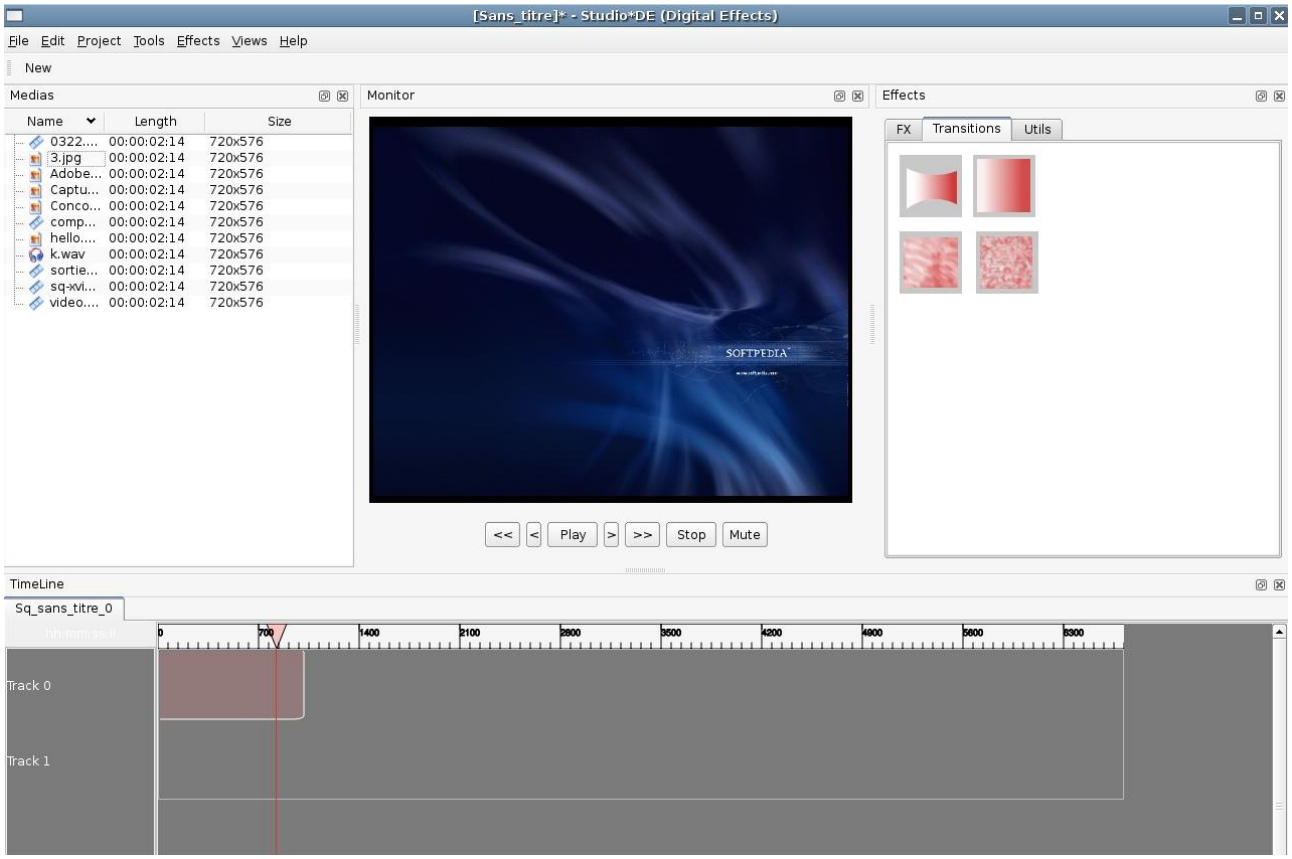

*Fig. 43 : L'interface principale de Studio*DE*

ORDONNANCEMENT DES TACHES

10/2006	> 10/2006	Définition du projet et choix des fonctionnalités de l'application à mettre en oeuvre

Guillaume

11/2006	> 11/2006	Inspection des logiciels de montage concurrents
12/2006	> 12/2006	Première analyse de la librairie GStreamer; Recherches sur l'organisation structure MVC et analyse de codes de logiciels Open Source
12/2006	> 01/2007	Implémentation d'une coquille vide d'interface, gestionnaire de Médias
01/2007	> 02/2007	Implémentation de la structure
02/2007	> 03/2007	Recherches sur les librairies vidéo, notamment la librairie AVIfile et le player VLC
03/2007	> 04/2007	Implémentation de la timeline
04/2007	> 05/2007	Gestion du drag and drop

Elica

11/2006	> 12/2006	Recherches sur le tracking
12/2007	> 12/2006	Implémentation du FullSearch
01/2007	> 01/2007	Recherches sur les algorithmes plus évolués
01/2007	> 02/2007	Implémentation de 3 autres algorithmes («spiral search», «greedy F» et «integral projections» avec optimisation)
02/2007	> 03/2007	Inspection des interfaces des logiciels de morphing existants
03/2007	> 04/2007	Implémentation de l'interface de l'effet « Morphing vidéo »
04/2007	> 05/2007	Implémentation de la gestion d'erreurs pour le tracking

Fabien

11/2006	> 11/2006	Recherches sur les différentes méthodes de morphing et inspection des logiciels de morphing existants et des résultats en fonction des techniques
11/2006	> 12/2006	Recherches sur la triangulation de Delaunay (librairie CGALS, puis choix de la libRT)
12/2006	> 02/2007	Implémentation du morphing
02/2007	> 03/2007	Recherches sur la détection des points clefs d'une image et implémentation de Harris
03/2007	> 04/2007	Recherche sur les plugins et implémentation d'une structure de plugin et d'une structure de chargement
04/2007	> 04/2007	Recherches sur les librairies vidéo
04/2007	> 05/2007	Implémentation de classes pour gérer la vidéo, basées sur la librairie GStreamer

DROIT

A) LIBRAIRIES

Notre projet est un projet étudiant qui se veut un logiciel Open Source sous licence GPL. Il utilise plusieurs librairies :

- Langage de programmation utilisé :

  C++: sous licence LGPL

- L'application étant écrite en C++, elle nécessite un compilateur :

  g++: licence GPL

- Interface graphique :

  Qt: sous double licence – une licence GPL gratuite pour les projets libres et une licence payante. Comme notre projet est un projet GPL, nous pouvons donc utiliser Qt librement.

- Librairie vidéo :

  GStreamer : sous licence LGPL. Son utilisation est donc totalement libre. La seule limitation de cette librairie est son portage sous Windows encore en développement (même s'il existe déjà une première version).

- Librairie de l'université de Marne-la-Vallée :

  libRT: librairie encore en développement, la partie utilisée dans cette application devrait être proposée sous une licence Open Source.

- Librairie pour la gestion des plugins :

  libdl: sous licence LGPL. La limitation de cette librairie est qu'elle n'est pas portable sous Windows.

- Librairie pour la gestion du format d'image JPG :

  libjpeg: sous licence BSD (compatible GPL)

B) PROJET

Etant un projet universitaire, ce projet ne peut pas être vendu. En revanche, nous restons propriétaires de notre code, et nous pouvons donc proposer celui-ci sous licence GPL.

PROMOTION

A) CHARTE GRAPHIQUE

La charte graphique correspond à l'ensemble des contraintes stylistiques et visuelles. Elle représente l'ensemble des codes graphiques, colorés et formels créés pour la communication visuelle de l'application. C'est à ce titre qu'ils doivent être particulièrement clairs et pertinents.

Ces règles sont appliquées aux déclinaisons du logo (dû aux différents supports possibles), aux mises en pages de documents, et du projet. Cette charte doit transmettre la cohérence de l'image de notre projet, c'est son identité visuelle.

Dans l'esprit des iconographes, le rouge représente l'incandescence et l'activité. Il est souvent associé à l'amour. Chargé du pouvoir régénérateur et purificateur, il peut également être lié au sang et à la flamme, au combat et à la colère, à la force et la vitalité. Le rouge est donc une couleur forte. Associé au blanc uni et pur, la puissance du rouge est accentuée.

Notre logo s'inscrit dans cette volonté de création d'un logiciel fort et robuste. Le logo est un "dé" en 3d. Il reprend ainsi la prononciation du nom du logiciel Studio*DE.

D'un autre côté l'aspect géométrique et mathématique du dé fait écho à la rigueur et à la qualité de la structure de l'application.

Et enfin, le but d'un dé est de jouer à un jeu de hasard. Le logiciel n'oublie donc pas que son but est de créer des films et qu'il ne doit pas faire disparaitre la partie créative. Des effets comme le tracking ne sont pas des effets rigoureux ou scientifiques mais sont des effets dans lesquels le hasard a toute sa place.

Enfin il ne faut pas oublier qu'un film est un loisir, alors il faut jouer !

B) SITE INTERNET

La promotion du projet se fera essentiellement sur Internet. Il dispose déjà d'un site dédié – http://studiode.free.fr – qui est en cours de développement mais dont les bases sont déjà posées.

Ce site propose d'abord le téléchargement du programme, ainsi qu'un accès à la documentation et au code (après inscription). Il hébergera également tous les plugins existants et à venir.

Ce sera en plus un espace dédié à la recherche liée au projet, où le visiteur aura accès, sous forme de références, liens et résumés, à tous les articles et travaux de recherche dans le domaine concerné qui ont servi de base ou de référence pour le projet, et qui permettent notamment d'approfondir le côté théorique de son fonctionnement.

Troisièmement, dans l'esprit d'un projet communautaire, tout utilisateur enregistré peut poster des avis et commentaires sur les articles ou bien émettre des idées, des propositions ou des bouts de code sur le forum.

Enfin, la partie privée du site, accessible pour l'instant aux administrateurs, propose des outils pour la gestion du projet, tels qu'un calendrier interactif des tâches.

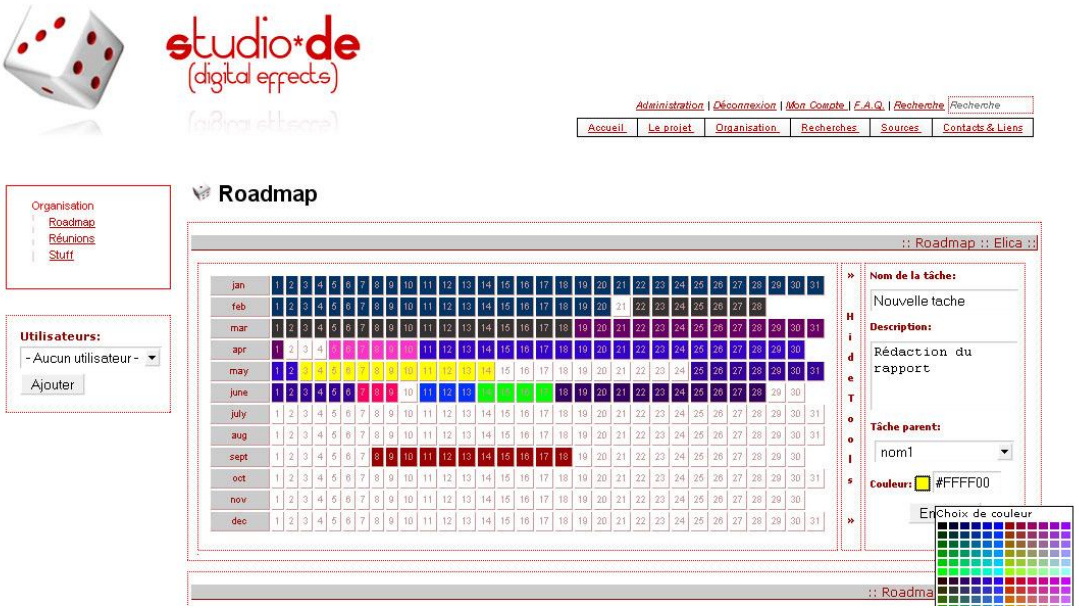

Fig. Le site internet studiode.free.fr – Roadmap avec outil de gestion des tâches

BILAN

Réunis près d'un an autour du même but, nous sommes aujourd'hui heureux d'avoir pu mener à bien le projet Studio*DE (Digital Effects). L'aventure se termine mais laisse ses traces. Tout au long du projet nous avons appris et tiré des enseignements, retenu des méthodes, ensemble. La diversité des profils, d'intérêts et de parcours de différents membres du groupe a été un avantage indéniable quant au brassage positif des idées.

L'intérêt d'un tel projet est de nous permettre de développer nos connaissance dans plusieurs domaines (programmation, mais également recherche), mais également de nous développer nous mêmes. En cela le travail de groupe est une expérience très enrichissante. Il est clair que le nombre de 3 personnes apporte des avantages concernant les problèmes d'organisation, et la gestion en est moins difficile que pour les plus grands groupes. En cela il ne nous a pas été très compliqué de nous réunir pour pouvoir établir des réunions. Nous avons su nous organiser et nous responsabiliser. Cependant, être trois impose une charge plus importante de travail par personne, et également une certaine discipline en matière de gestion de la ressource temporelle (en effet, à avoir des modules plus importantes et plus indépendants, les individus dépendent moins les uns des autres pendant une certaine partie du projet. Cette partie est à considérer car c'est là que les retards accumulés pourront avoir des conséquences lors des regroupements des implémentations). Dans un projet tel que celui-ci, il ne faut donc pas se démotiver et rester constamment avec ses objectifs en tête, motiver également le reste du groupe pour avancer sans prendre de retard. Une cause des retards sont également les « imprévus » qu'il faut rendre en considération dès le début du projet, lors de la réalisation des plannings initiaux. En effet, on ne peut préjuger à l'avance de tel ou tel événement, qu'il soit lié à la formation (devoir prioritaire) ou non, qu'il soit lié au projet (problème à un certain stade de développement) ou non. Donc il faut prévoir une part de doute dans les durées imposées initialement pour ne pas dépasser la durée critique d'une tâche. Nous avons appris au fur et à mesure à gérer notre temps, à être réalistes sur nos objectifs et nos capacités. Ce projet a en effet été l'occasion d'un accomplissement personnel où chacun doit trouver sa place dans un groupe.

En conclusion, nous avons été une équipe engagée pendant près de 8 mois dans son projet. S'imposant de nombreuses règles et organisant le plus de réunions possibles entre nous, nous n'avons eu de cesse de communiquer pour sans cesse avoir la vision la plus globale du projet. Ce projet fut une expérience humaine et professionnelle extrêmement enrichissante. Chacun s'implique pou un projet commun : la réalisation d'une application qui nous tient à cœur. Nous sommes donc fiers aujourd'hui de pouvoir présenter Studio*DE (Digital Effects).

BIBLIOGRAPHIE & REFERENCES

A) GENERALE

- **Thèse de Laurent Bonnaud**, qui présente des techniques pour réaliser du suivit de mouvement, base de la détection d'objets, interpolation temporelle.
  http://www.irisa.fr/centredoc/publis/theses/1998/irisapublication.2006-01-20.0430185938

- Charlier P. (2000), **Hypermédias et expérience d'apprentissage**. Thèse de doctorat en sciences sociales (information et communication), Louvain-la-Neuve, UCL, Faculté des sciences économiques, sociales et politiques, Département de communication, 395 p.

B) MORPHING

- **Triangulation de Delaunay**, implémentée dans la librairie CGAL
  http://www.cgal.org/

- De nombreuses publications sur **la triangulation de Delaunay** et le diagramme de Voronoi
  http://www.igi.tugraz.at/auren/

C) POINTS DE HARRIS

- Explications de **l'algorithme de Harris** :
  http://en.wikipedia.org/wiki/Corner_detection#The_Harris_.26_Stephens_.2F_Plessey_corner_detection_algorithm

- Une **explication simple** de sa mise en œuvre :
  http://devernay.free.fr/cours/vision/iicao/td5.html

- Extension de **l'algorithme de Harris en RGB** :
  http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MONTESSINOS/node6.html

D) TRACKING

- **Video compression** explained
  http://newmediarepublic.com/dvideo/compression.html

- **Search Algorithms** for Block-Matching in Motion Estimation
  http://www.ece.cmu.edu/~ee899/project/deepak_mid.htm

- **Le suivi de blobs** comme base pour la caractérisation du mouvement dans des séquences audiovisuelles
  http://rfv.insa-lyon.fr/~megret/papers/RFV-RR-2001-02.pdf

- ORPHEUS: A new system for **semi-automatic building extraction**
  http://www.ipf.tuwien.ac.at/fr/buildings/diss/node5.html

E) PLUGINS

- Documentation du standard de plugins d'effets vidéo **Open FX**
  http://openfx.sourceforge.net/

F) VIDEO

- Le site officiel de **GStreamer** :
  http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer/html/index.html

- La documentation sur les **plugins de GStreamer** :
  http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gst-plugins-base-plugins/html/ch02.html

- **Documentation en français** (par les personnes du projet PiTivi) :
  http://www.bilboed.com/projet/Traduction%20Manuel%20GStreamer.pdf

- L'explication du **pipeline** :
  http://www.cin.ufpe.br/~cinlug/wiki/index.php/Introducing_GStreamer

- Une **conférence** par un développeur de GStreamer :
  http://www.bestechvideos.com/2007/03/13/linuxconfau-gstreamer-plugin-development/

G) QT & INTERFACE

- Deux **sites communautaires** de référence autour de Qt
  http://www.qtfr.org/
  http://blog.qtfr.org/

- La **documentation officielle de QT** 4.2.2
  http://doc.trolltech.com/4.2/index.html

- **C++ GUI Programming With Qt 4,** Jasmin Blanchette (Auteur), Mark Summerfield (Auteur)

- Un des **projets qui nous ont aidés**, notamment concernant la structure de développement – **kdenlive**. C'est un projet d'édition vidéo non linéaire pour Linux. Développé pour KDE et basé sur MLT (framework multimédia), il est encore en développement mais propose de nombreuses fonctionnalités
  http://kdenlive.org/

- Un **autre projet intéressant – ktoon**. Il est développé pour l'environnement KDE et propose un service d'animation 2D
  http://ktoon.toonka.com/documentation/index.php?title=Documentation_en_Fran%E7ais

H) Droit

- **Les licences GPL** :
  http://www.gnu.org/licenses/gpl-howto.fr.html

ANNEXES

LISTE DES ANNEXES:

A) CAHIER DES CHARGES COMMANDITAIRE

B) CARNET DE BORD DES REUNIONS

C) GUIDE D'UTILISATION